Autómatas y Lenguajes

Práctica 2 - Análisis Sintáctico

Material suministrado

grammar	
grammar.py	Único fichero a entregar
LL1Table	Clase
analyze()	Método a modificar en el **ejercicio 2**
Grammar	Clase
compute_first()	Método a modificar en el **ejercicio 3**
compute_follow()	Método a modificar en el **ejercicio 4**
get_ll1_table()	Método a modificar en el **ejercicio 5**
test_analyze.py	Fichero con tests (¡realizad tests adicionales!)
test_first.py	Fichero con tests (¡realizad tests adicionales!)
test_follow.py	Fichero con tests (¡realizad tests adicionales!)
utils.py	No se modifica ni entrega

Ejercicio 2

En `grammar.py`

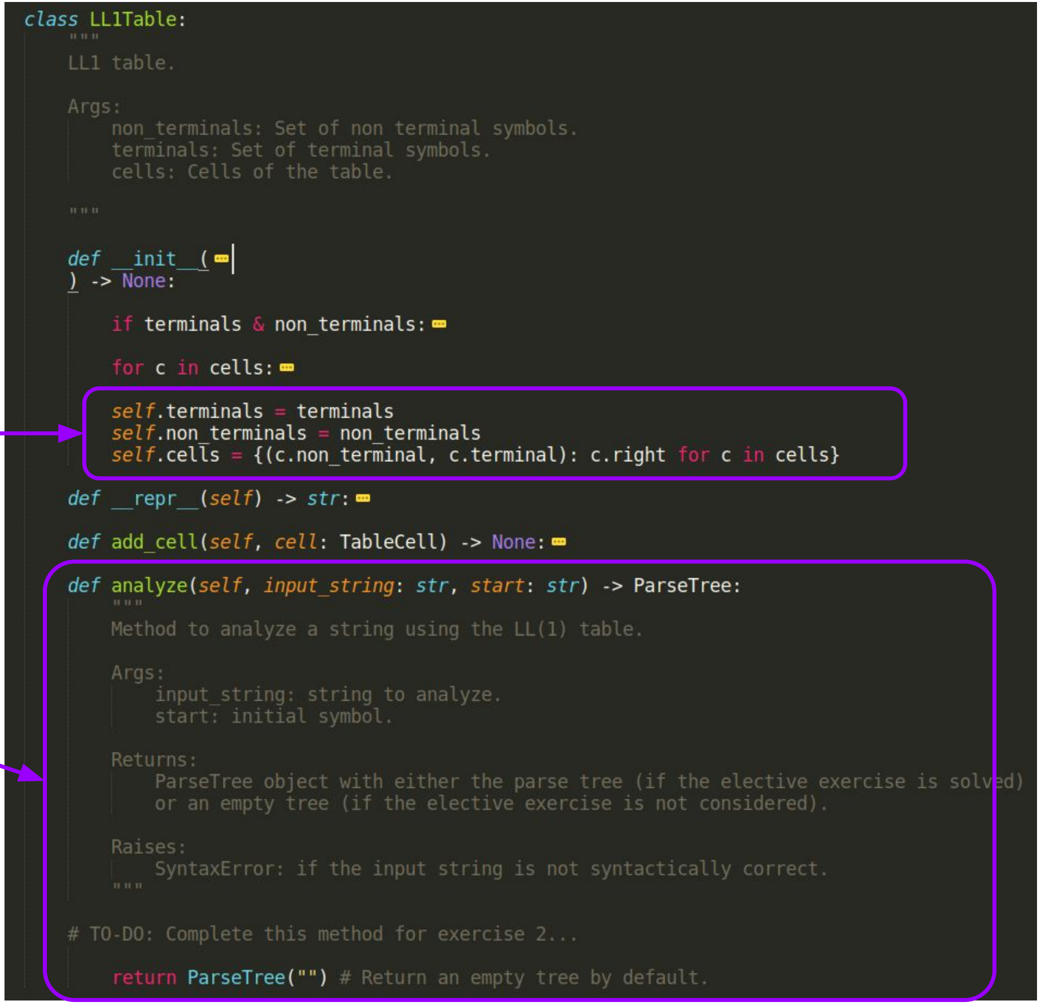
`-> En LL1Table`

La tabla contiene estos atributos

Completar

Páginas 38-49 de la presentación

Algoritmo en página 46

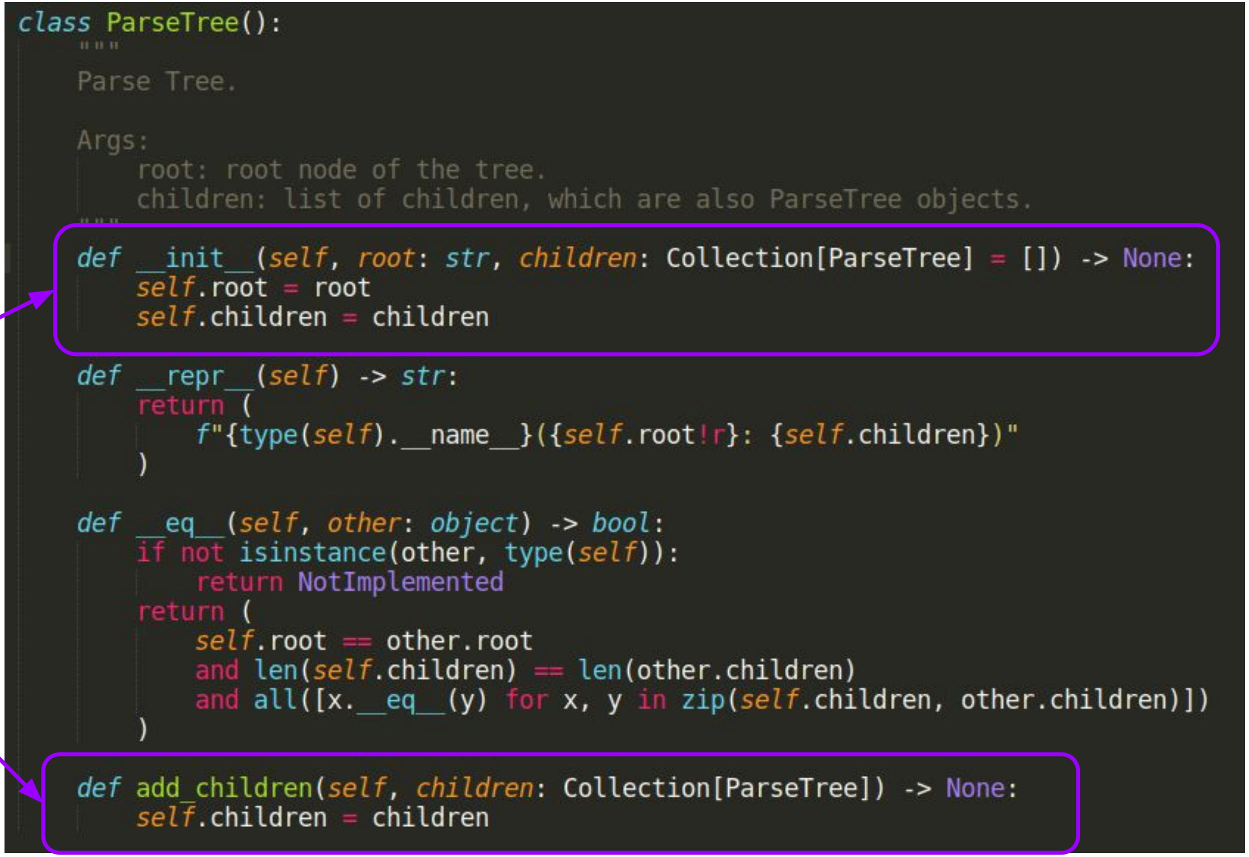


```
class LL1Table:
    """
    LL1 table.

    Args:
        non_terminals: Set of non terminal symbols.
        terminals: Set of terminal symbols.
        cells: Cells of the table.

    """

    def __init__(⋯
    ) -> None:

        if terminals & non_terminals:⋯

        for c in cells:⋯

        self.terminals = terminals
        self.non_terminals = non_terminals
        self.cells = {(c.non_terminal, c.terminal): c.right for c in cells}

    def __repr__(self) -> str:⋯

    def add_cell(self, cell: TableCell) -> None:⋯

    def analyze(self, input_string: str, start: str) -> ParseTree:
        """
        Method to analyze a string using the LL(1) table.

        Args:
            input_string: string to analyze.
            start: initial symbol.

        Returns:
            ParseTree object with either the parse tree (if the elective exercise is solved)
            or an empty tree (if the elective exercise is not considered).

        Raises:
            SyntaxError: if the input string is not syntactically correct.
        """

    # TO-DO: Complete this method for exercise 2...

        return ParseTree("") # Return an empty tree by default.
```

Ejercicio 2

En `grammar.py`
  `-> En ParseTree`

Tened en cuenta que tenéis esto



```python
class ParseTree():
    """
    Parse Tree.

    Args:
        root: root node of the tree.
        children: list of children, which are also ParseTree objects.
    """
    def __init__(self, root: str, children: Collection[ParseTree] = []) -> None:
        self.root = root
        self.children = children

    def __repr__(self) -> str:
        return (
            f"{type(self).__name__}({self.root!r}: {self.children})"
        )

    def __eq__(self, other: object) -> bool:
        if not isinstance(other, type(self)):
            return NotImplemented
        return (
            self.root == other.root
            and len(self.children) == len(other.children)
            and all([x.__eq__(y) for x, y in zip(self.children, other.children)])
        )

    def add_children(self, children: Collection[ParseTree]) -> None:
        self.children = children
```

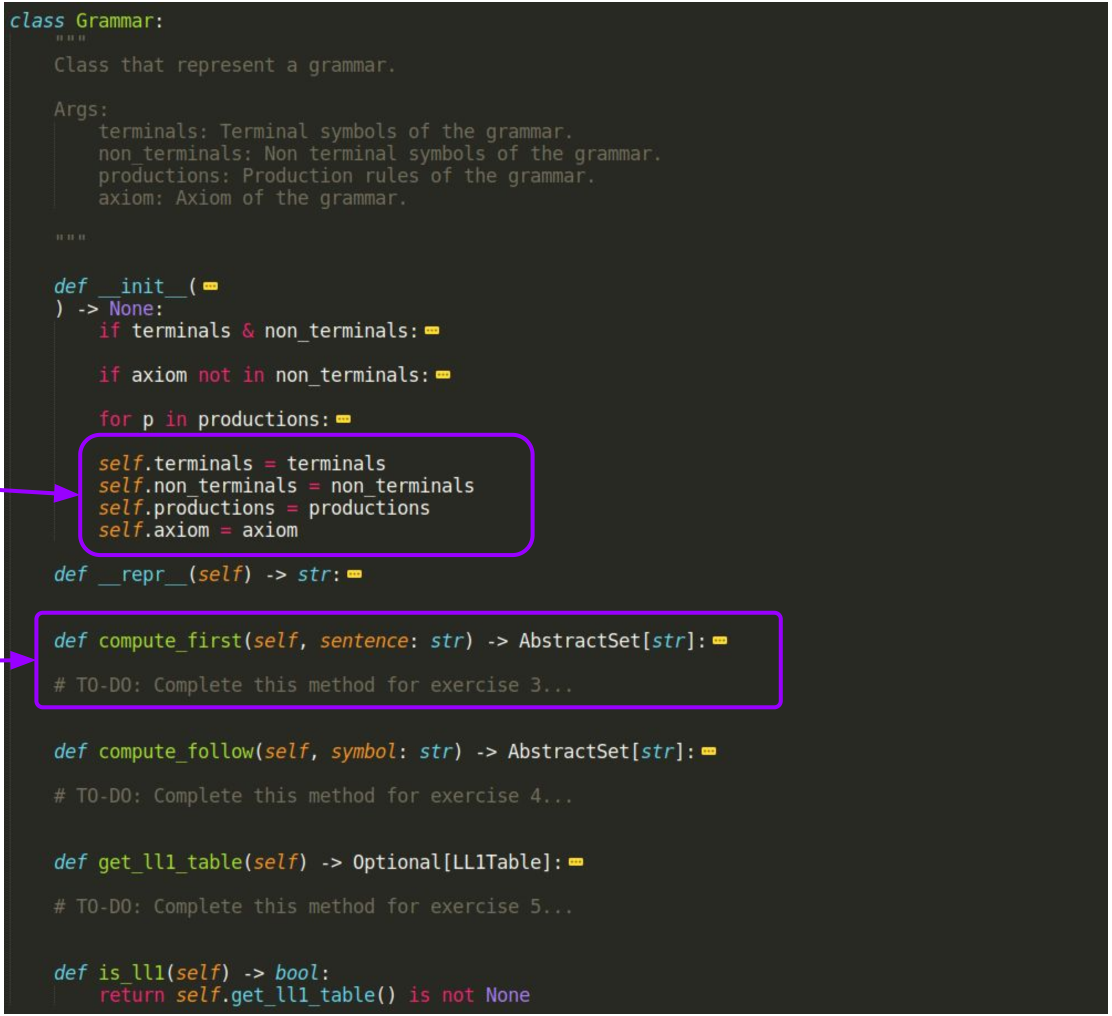
Ejercicio 3

En `grammar.py`
`-> En LL1Table`

La gramática contiene estos atributos

Completar

Páginas 52-54 de la presentación
Algoritmo en página 53



```
class Grammar:
    """
    Class that represent a grammar.

    Args:
        terminals: Terminal symbols of the grammar.
        non_terminals: Non terminal symbols of the grammar.
        productions: Production rules of the grammar.
        axiom: Axiom of the grammar.

    """

    def __init__(⋯
    ) -> None:
        if terminals & non_terminals:⋯

        if axiom not in non_terminals:⋯

        for p in productions:⋯

        self.terminals = terminals
        self.non_terminals = non_terminals
        self.productions = productions
        self.axiom = axiom

    def __repr__(self) -> str:⋯

    def compute_first(self, sentence: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 3...

    def compute_follow(self, symbol: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 4...

    def get_ll1_table(self) -> Optional[LL1Table]:⋯

    # TO-DO: Complete this method for exercise 5...

    def is_ll1(self) -> bool:
        return self.get_ll1_table() is not None
```

Ejercicio 4

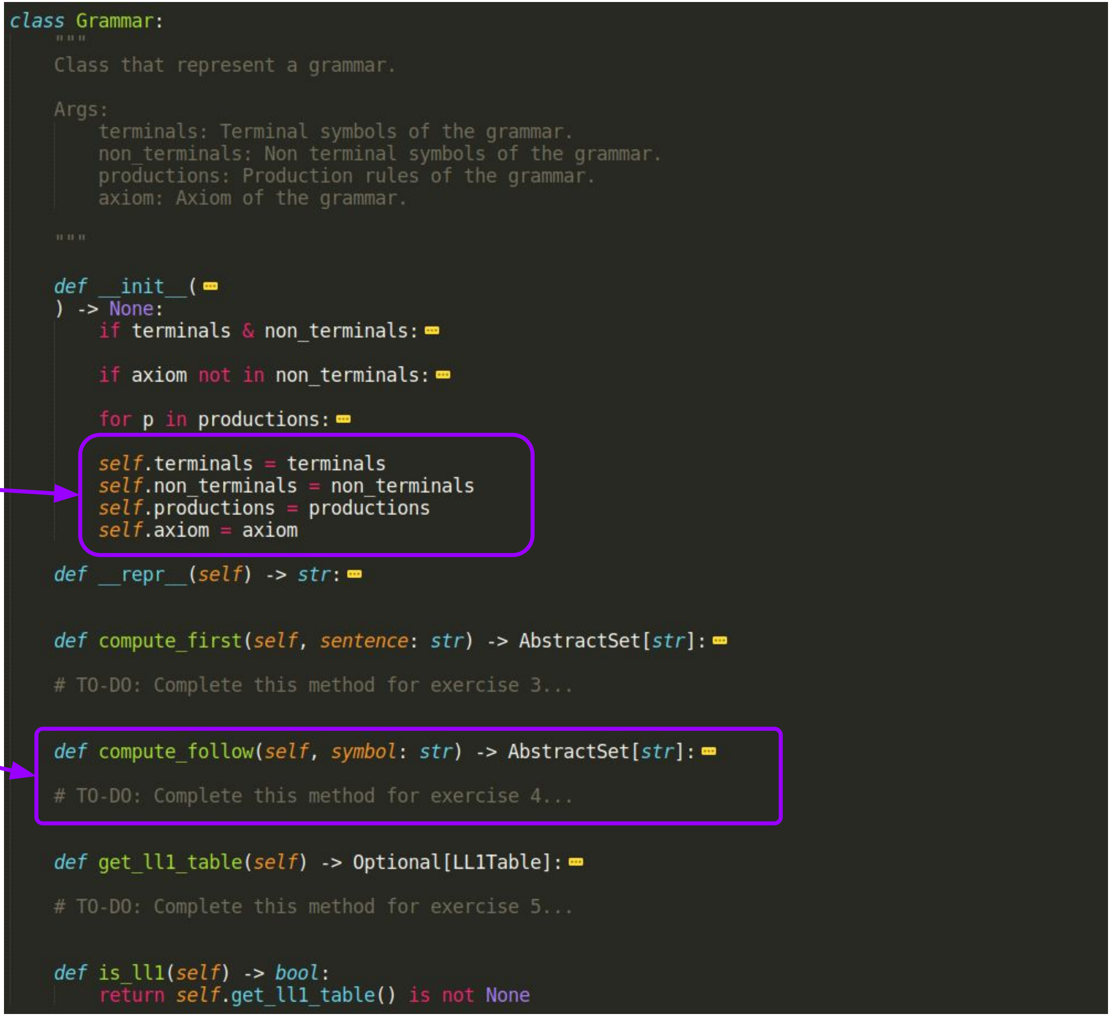
En `grammar.py`

`-> En LL1Table`

La gramática contiene estos atributos

Completar

Páginas 52,55-57 de la presentación *Algoritmo en página 56*



```python
class Grammar:
    """
    Class that represent a grammar.

    Args:
        terminals: Terminal symbols of the grammar.
        non_terminals: Non terminal symbols of the grammar.
        productions: Production rules of the grammar.
        axiom: Axiom of the grammar.

    """

    def __init__(⋯
    ) -> None:
        if terminals & non_terminals:⋯

        if axiom not in non_terminals:⋯

        for p in productions:⋯

        self.terminals = terminals
        self.non_terminals = non_terminals
        self.productions = productions
        self.axiom = axiom

    def __repr__(self) -> str:⋯


    def compute_first(self, sentence: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 3...

    def compute_follow(self, symbol: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 4...

    def get_ll1_table(self) -> Optional[LL1Table]:⋯

    # TO-DO: Complete this method for exercise 5...

    def is_ll1(self) -> bool:
        return self.get_ll1_table() is not None
```

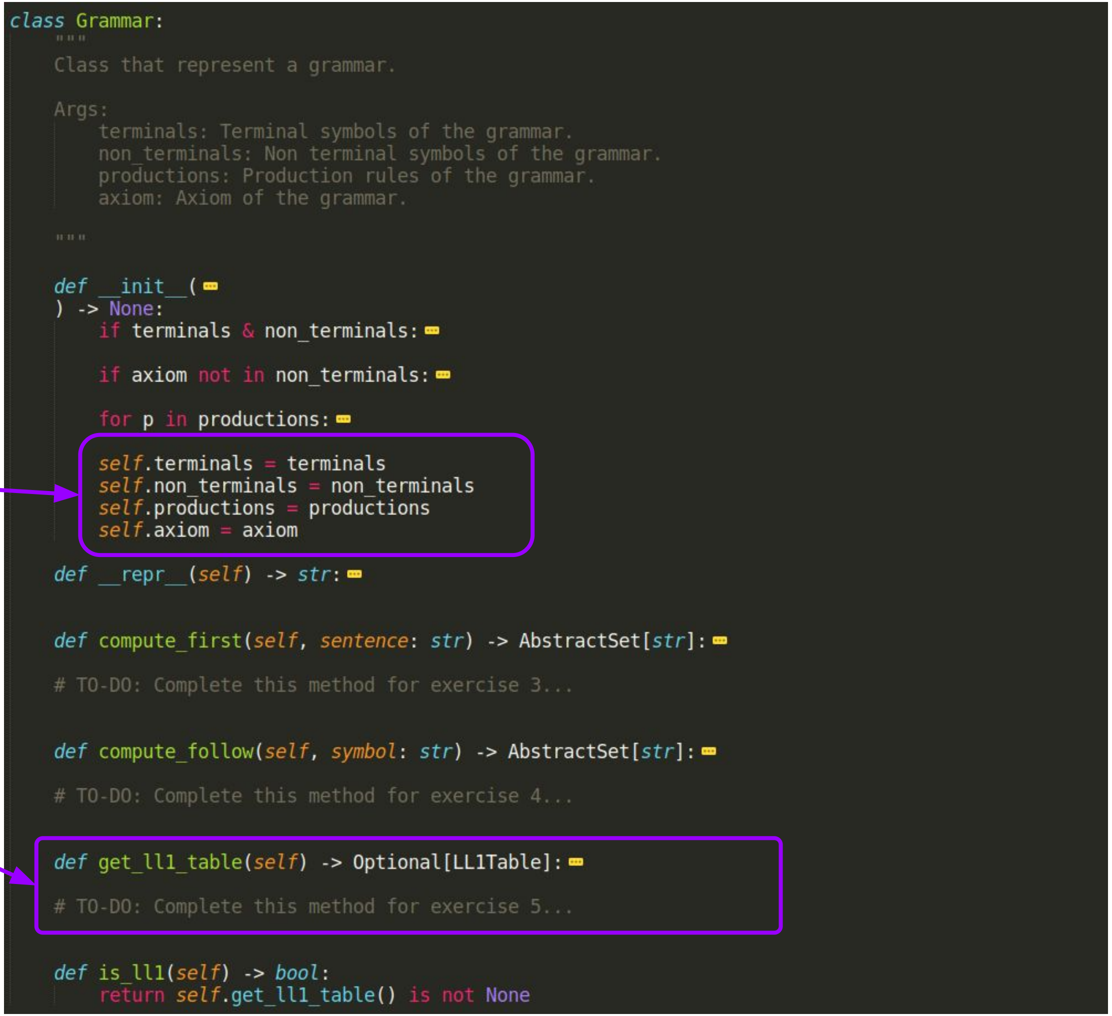
Ejercicio 4

En `grammar.py`
  `-> En LL1Table`

La gramática contiene estos atributos

Completar

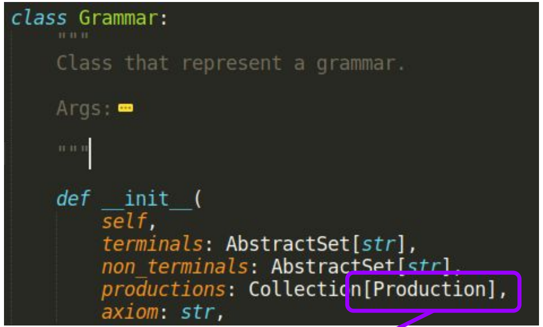
Páginas 50-51 de la presentación *Algoritmo en página 51*



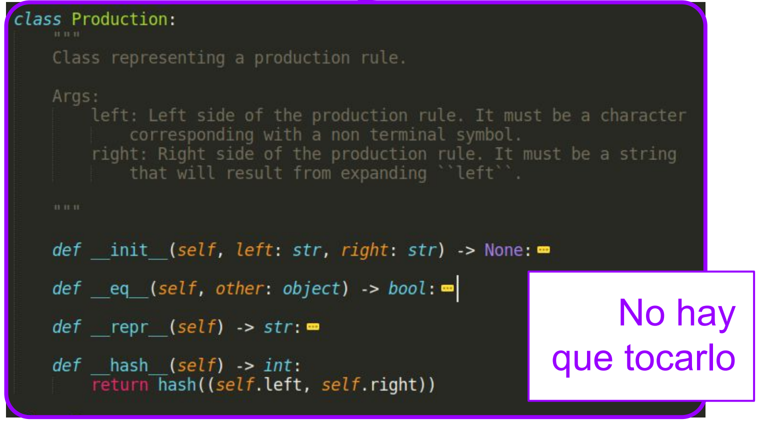
```
class Grammar:
    """
    Class that represent a grammar.

    Args:
        terminals: Terminal symbols of the grammar.
        non_terminals: Non terminal symbols of the grammar.
        productions: Production rules of the grammar.
        axiom: Axiom of the grammar.

    """

    def __init__(⋯
    ) -> None:
        if terminals & non_terminals:⋯

        if axiom not in non_terminals:⋯

        for p in productions:⋯

        self.terminals = terminals
        self.non_terminals = non_terminals
        self.productions = productions
        self.axiom = axiom

    def __repr__(self) -> str:⋯


    def compute_first(self, sentence: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 3...


    def compute_follow(self, symbol: str) -> AbstractSet[str]:⋯

    # TO-DO: Complete this method for exercise 4...


    def get_ll1_table(self) -> Optional[LL1Table]:⋯

    # TO-DO: Complete this method for exercise 5...


    def is_ll1(self) -> bool:
        return self.get_ll1_table() is not None
```

Ejercicios 3-5

No tenéis que crear objetos `Production`, pero sí usarlo cuando es atributo

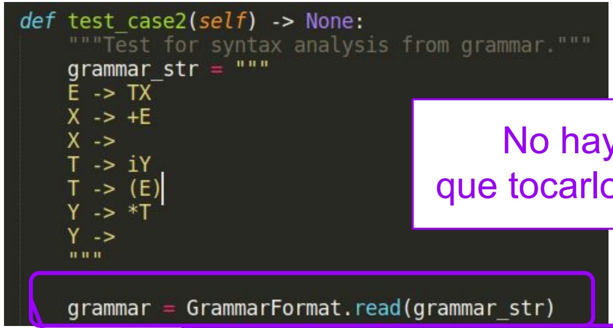


```
class Grammar:
    """
    Class that represent a grammar.

    Args: ...

    """

    def __init__(
        self,
        terminals: AbstractSet[str],
        non_terminals: AbstractSet[str],
        productions: Collection[Production],
        axiom: str,
```



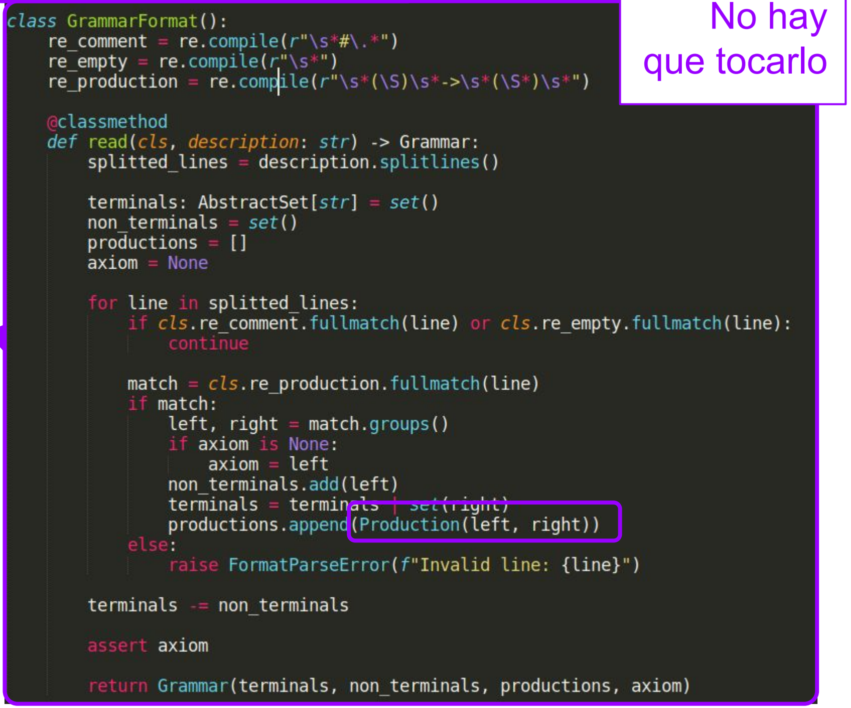
```
class Production:
    """
    Class representing a production rule.

    Args:
        left: Left side of the production rule. It must be a character
            corresponding with a non terminal symbol.
        right: Right side of the production rule. It must be a string
            that will result from expanding ``left``.

    """

    def __init__(self, left: str, right: str) -> None: ...

    def __eq__(self, other: object) -> bool: ...

    def __repr__(self) -> str: ...

    def __hash__(self) -> int:
        return hash((self.left, self.right))
```

No hay que tocarlo



```
def test_case2(self) -> None:
    """Test for syntax analysis from grammar."""
    grammar_str = """
    E -> TX
    X -> +E
    X ->
    T -> iY
    T -> (E)
    Y -> *T
    Y ->
    """

    grammar = GrammarFormat.read(grammar_str)
```

No hay que tocarlo



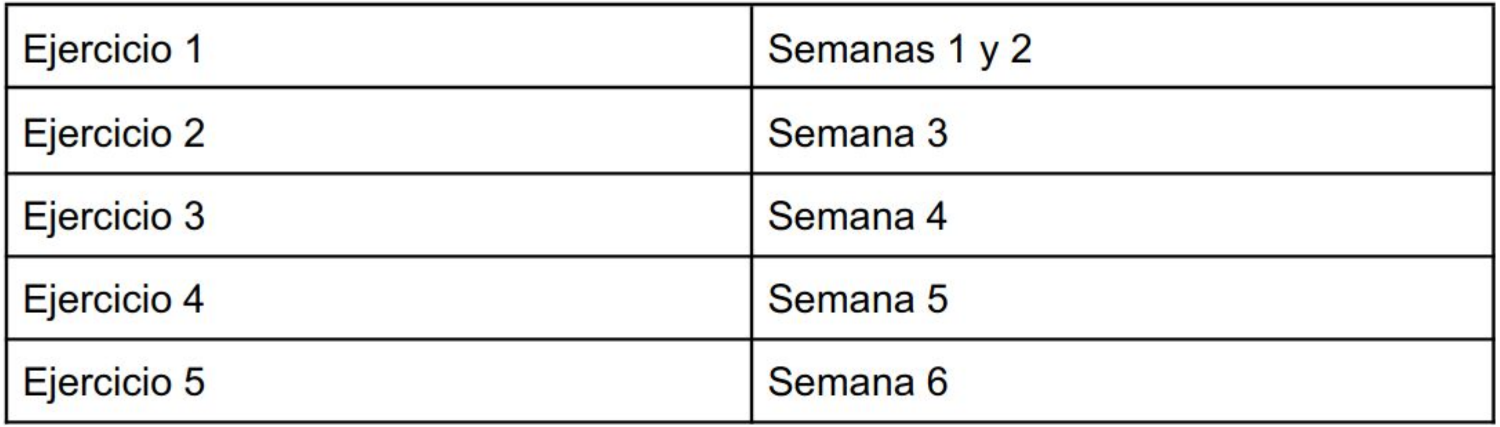
```
class GrammarFormat():
    re_comment = re.compile(r"\s*#\.*")
    re_empty = re.compile(r"\s*")
    re_production = re.compile(r"\s*(\S)\s*->\s*(\S*)\s*")

    @classmethod
    def read(cls, description: str) -> Grammar:
        splitted_lines = description.splitlines()

        terminals: AbstractSet[str] = set()
        non_terminals = set()
        productions = []
        axiom = None

        for line in splitted_lines:
            if cls.re_comment.fullmatch(line) or cls.re_empty.fullmatch(line):
                continue

            match = cls.re_production.fullmatch(line)
            if match:
                left, right = match.groups()
                if axiom is None:
                    axiom = left
                non_terminals.add(left)
                terminals = terminals | set(right)
                productions.append(Production(left, right))
            else:
                raise FormatParseError(f"Invalid line: {line}")

        terminals -= non_terminals

        assert axiom

        return Grammar(terminals, non_terminals, productions, axiom)
```

No hay que tocarlo

Quizá sea útil….

https://www.jflap.org/

Planificación



Ejercicio 1	Semanas 1 y 2
Ejercicio 2	Semana 3
Ejercicio 3	Semana 4
Ejercicio 4	Semana 5
Ejercicio 5	Semana 6