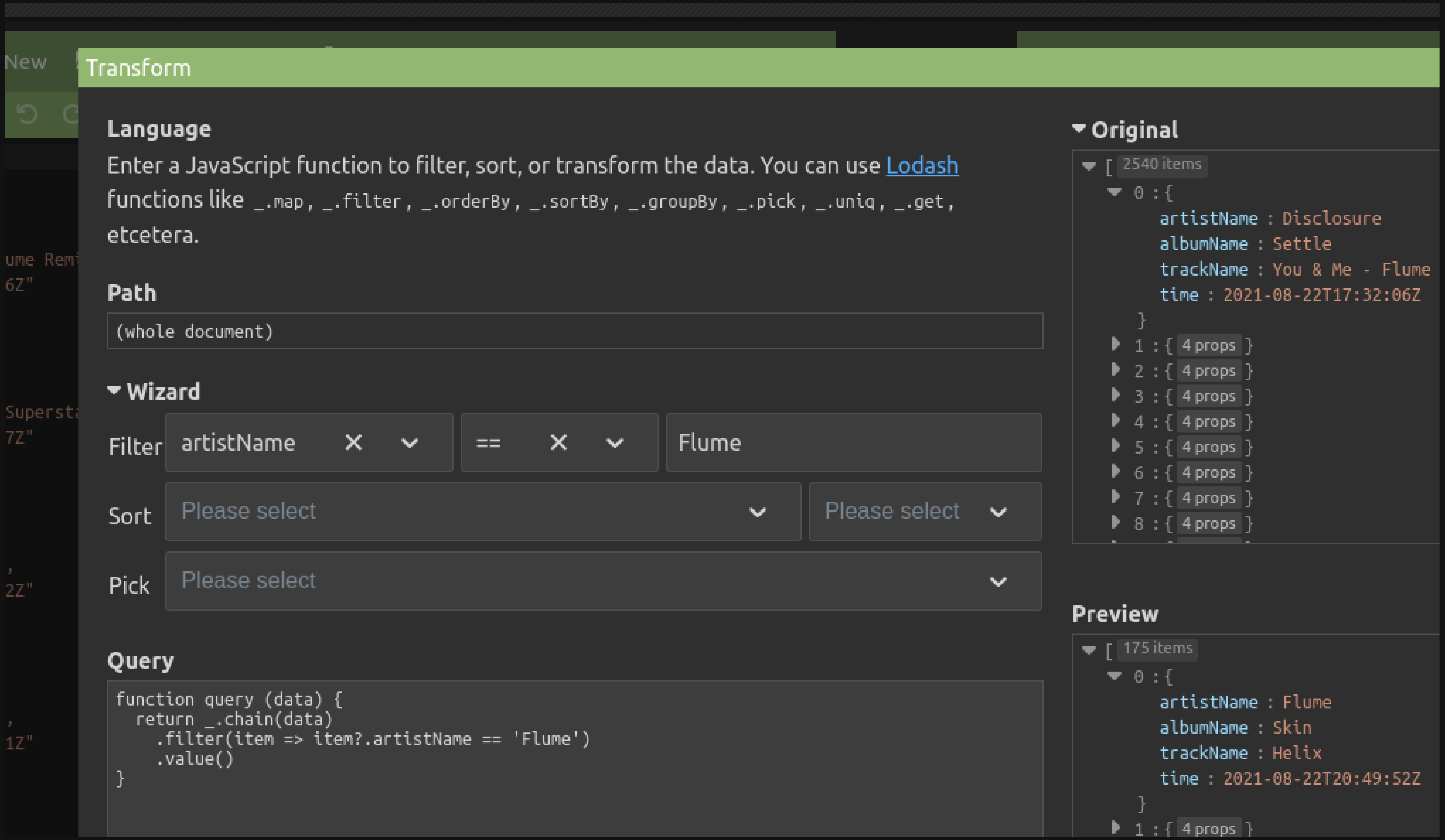
BQL: filtrando archivos CSV con lógica proposicional

Jesus Stevan Diaz Ingol

PROBLEMA


Transform

Language

Enter a JavaScript function to filter, sort, or transform the data. You can use Lodash functions like `_.map`, `_.filter`, `_.orderBy`, `_.sortBy`, `_.groupBy`, `_.pick`, `_.uniq`, `_.get`, etcetera.

Path

(whole document)

Wizard

Filter: artistName == Flume

Sort: Please select / Please select

Pick: Please select

Query

```
function query (data) {
  return _.chain(data)
    .filter(item => item?.artistName == 'Flume')
    .value()
}
```

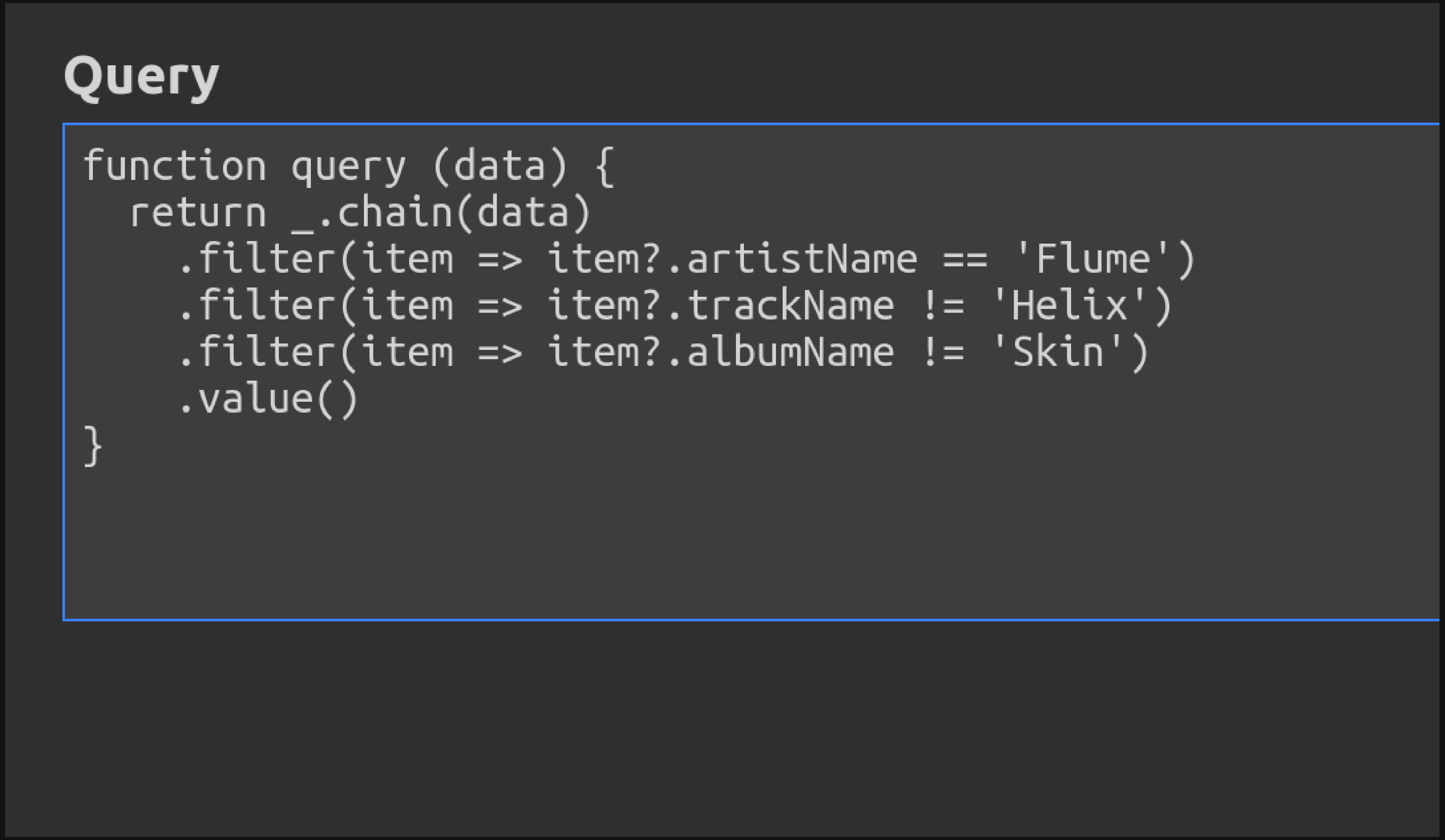
Original

```
[ 2540 items
  0 : {
      artistName : Disclosure
      albumName : Settle
      trackName : You & Me - Flume
      time : 2021-08-22T17:32:06Z
  }
  1 : { 4 props }
  2 : { 4 props }
  3 : { 4 props }
  4 : { 4 props }
  5 : { 4 props }
  6 : { 4 props }
  7 : { 4 props }
  8 : { 4 props }
```

Preview

```
[ 175 items
  0 : {
      artistName : Flume
      albumName : Skin
      trackName : Helix
      time : 2021-08-22T20:49:52Z
  }
  1 : { 4 props }
```

- Existen distintas herramientas para filtrar registros en archivos JSON y CSV de acuerdo al valor que asumen determinados **filtros respecto al contenido** de los registros.
- Estas herramientas recorren registro por registro, evaluan el filtro con los contenidos del registro actual, y dependiendo del resultado de dicha evaluación el registro será incluido o excluido.



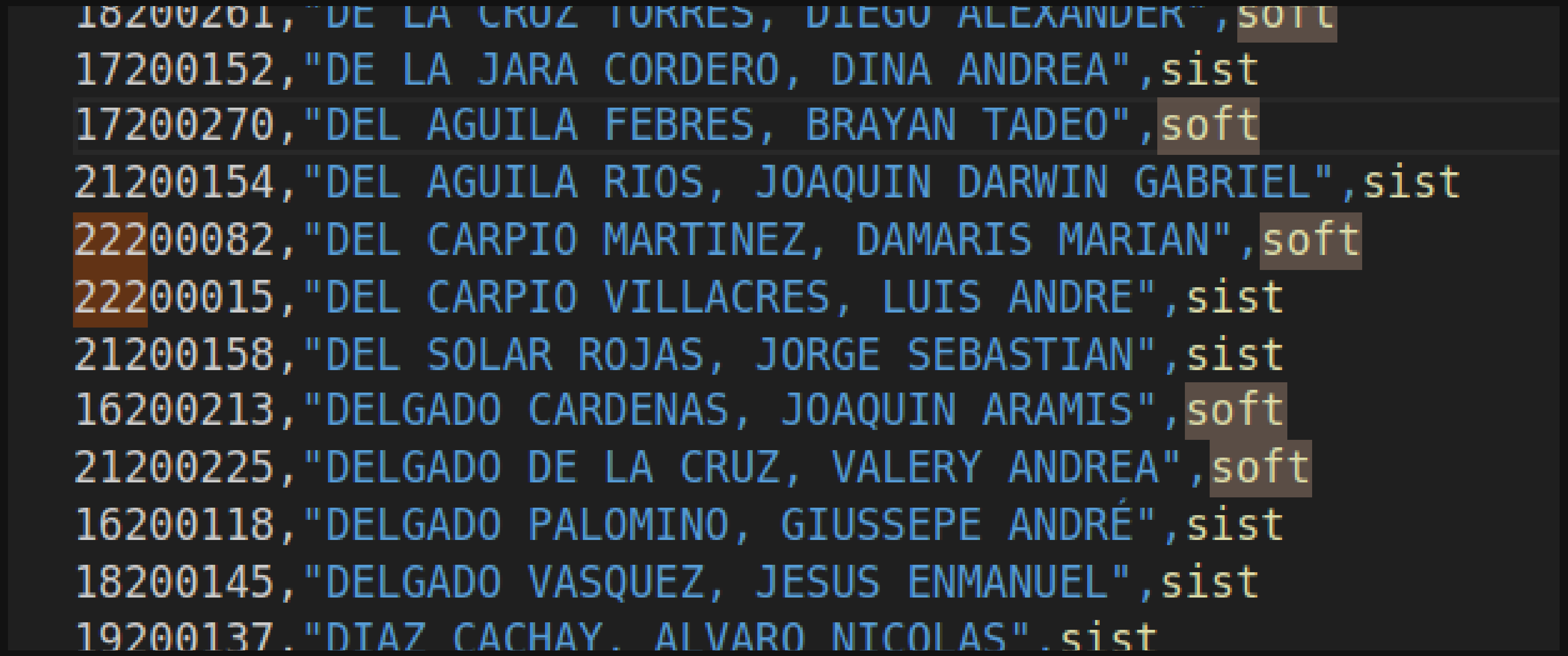
Query

```
function query (data) {
  return _.chain(data)
    .filter(item => item?.artistName == 'Flume')
    .filter(item => item?.trackName != 'Helix')
    .filter(item => item?.albumName != 'Skin')
    .value()
}
```

- Estos filtros son proposiciones que comparan campos del archivo con valores introducidos por el usuario.

- Si bien la mayoría de herramientas te permiten combinar filtros apilando uno sobre otro (conjunción), **pocas te permiten establecer relaciones lógicas más complejas**.

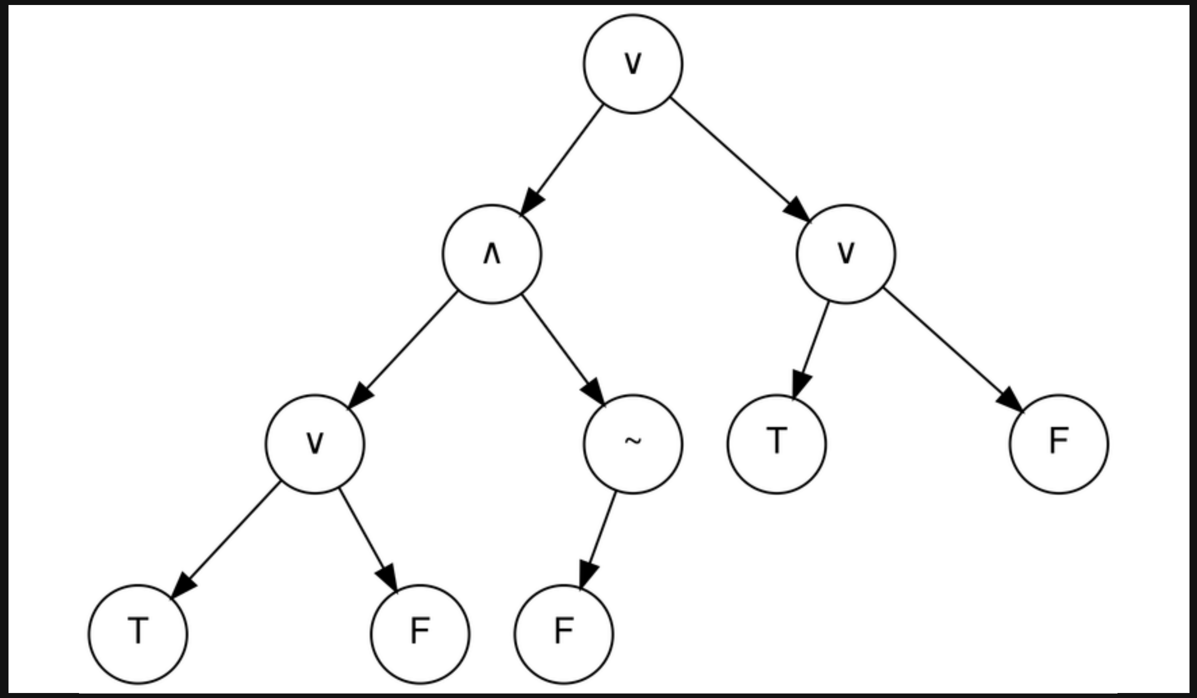
OBJETIVO GENERAL

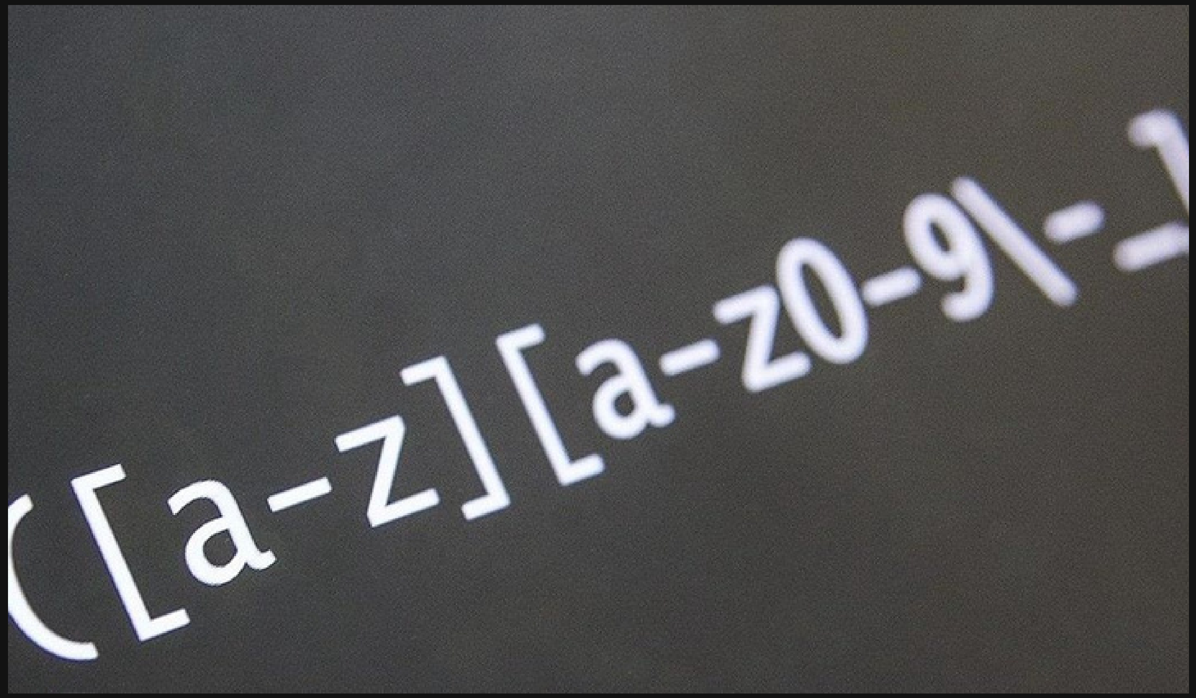


```
18200261,"DE LA CRUZ TORRES, DIEGO ALEXANDER",soft
17200152,"DE LA JARA CORDERO, DINA ANDREA",sist
17200270,"DEL AGUILA FEBRES, BRAYAN TADEO",soft
21200154,"DEL AGUILA RIOS, JOAQUIN DARWIN GABRIEL",sist
22200082,"DEL CARPIO MARTINEZ, DAMARIS MARIAN",soft
22200015,"DEL CARPIO VILLACRES, LUIS ANDRE",sist
21200158,"DEL SOLAR ROJAS, JORGE SEBASTIAN",sist
16200213,"DELGADO CARDENAS, JOAQUIN ARAMIS",soft
21200225,"DELGADO DE LA CRUZ, VALERY ANDREA",soft
16200118,"DELGADO PALOMINO, GIUSSEPE ANDRÉ",sist
18200145,"DELGADO VASQUEZ, JESUS ENMANUEL",sist
19200137,"DIAZ CACHAY, ALVARO NICOLAS",sist
```

Desarrollar una aplicación que permita filtrar registros en archivos CSV, según condiciones lógicas complejas introducidas por el usuario, que evalúan el contenido de dichos registros.

OBJETIVOS ESPECÍFICOS

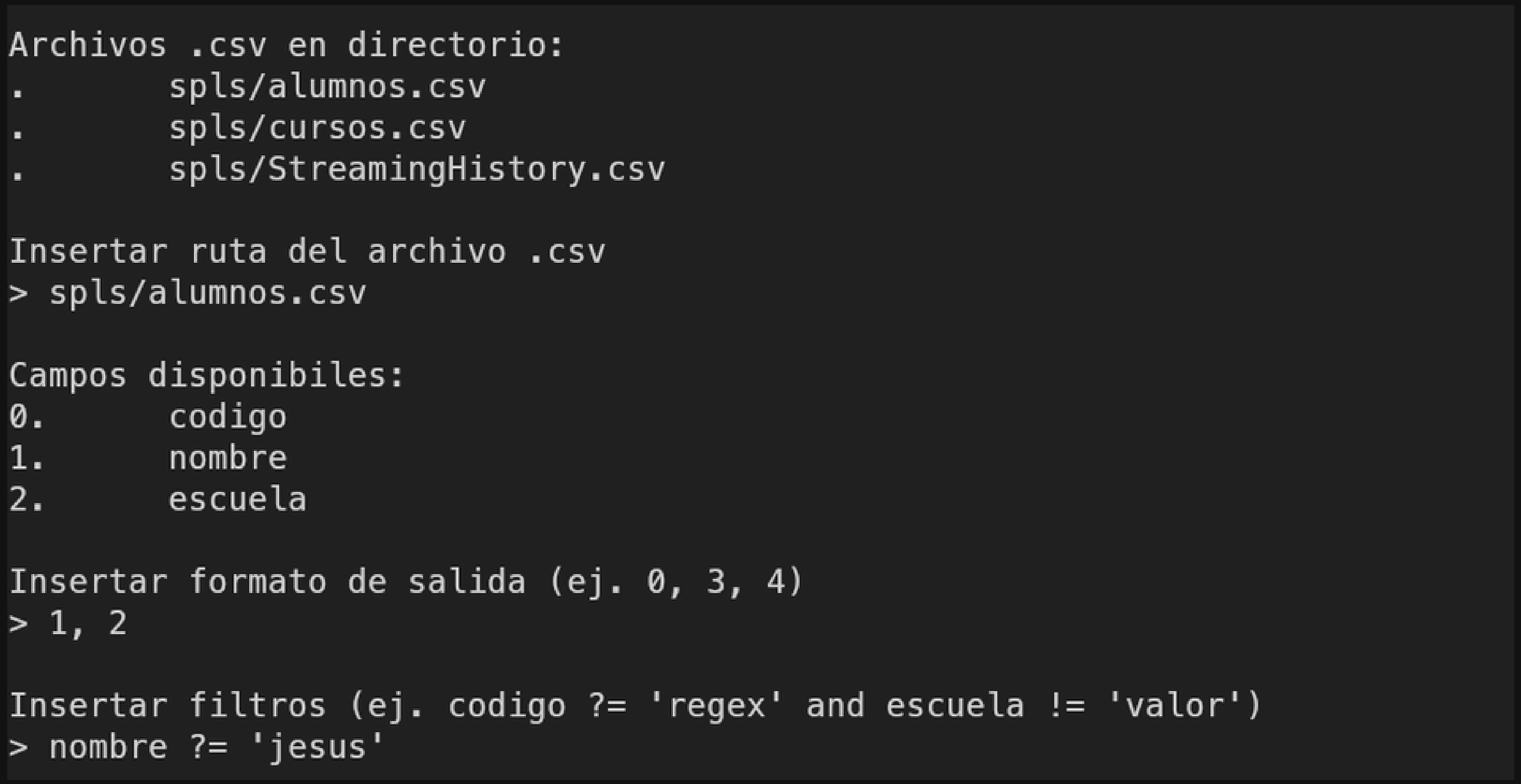




Aplicar los conceptos de lógica booleana al *parsing* y posterior evaluación de proposiciones lógicas en forma de cadena.

Aplicar los conceptos de expresiones regulares para evaluar condiciones de igualdad y similitud en registros.

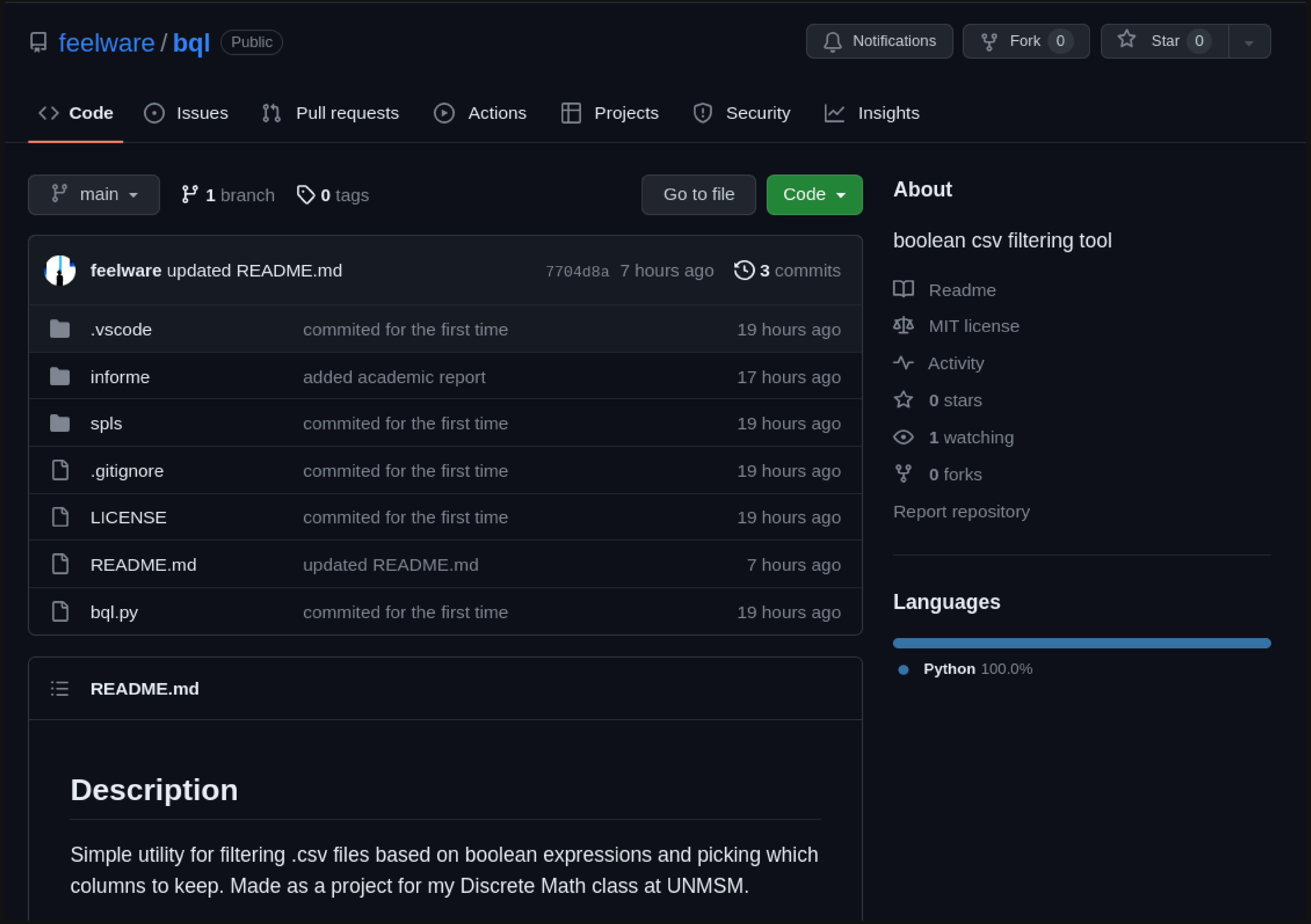
OBJETIVOS ESPECÍFICOS



```
Archivos .csv en directorio:
.       spls/alumnos.csv
.       spls/cursos.csv
.       spls/StreamingHistory.csv

Insertar ruta del archivo .csv
> spls/alumnos.csv

Campos disponibiles:
0.      codigo
1.      nombre
2.      escuela

Insertar formato de salida (ej. 0, 3, 4)
> 1, 2

Insertar filtros (ej. codigo ?= 'regex' and escuela != 'valor')
> nombre ?= 'jesus'
```

Desarrollar una interfaz de usuario que facilite la introducción de datos, que permita seleccionar qué campos devolver y en qué orden, así como elegir si se desea guardar el resultado como un nuevo archivo CSV.

PRODUCTO



feelware / bql Public

Notifications | Fork 0 | Star 0

Code | Issues | Pull requests | Actions | Projects | Security | Insights

main | 1 branch | 0 tags | Go to file | Code

feelware updated README.md 7704d8a 7 hours ago 3 commits

.vscode	commited for the first time	19 hours ago
informe	added academic report	17 hours ago
spls	commited for the first time	19 hours ago
.gitignore	commited for the first time	19 hours ago
LICENSE	commited for the first time	19 hours ago
README.md	updated README.md	7 hours ago
bql.py	commited for the first time	19 hours ago

README.md

Description

Simple utility for filtering .csv files based on boolean expressions and picking which columns to keep. Made as a project for my Discrete Math class at UNMSM.

About

boolean csv filtering tool

Readme
MIT license
Activity
0 stars
1 watching
0 forks
Report repository

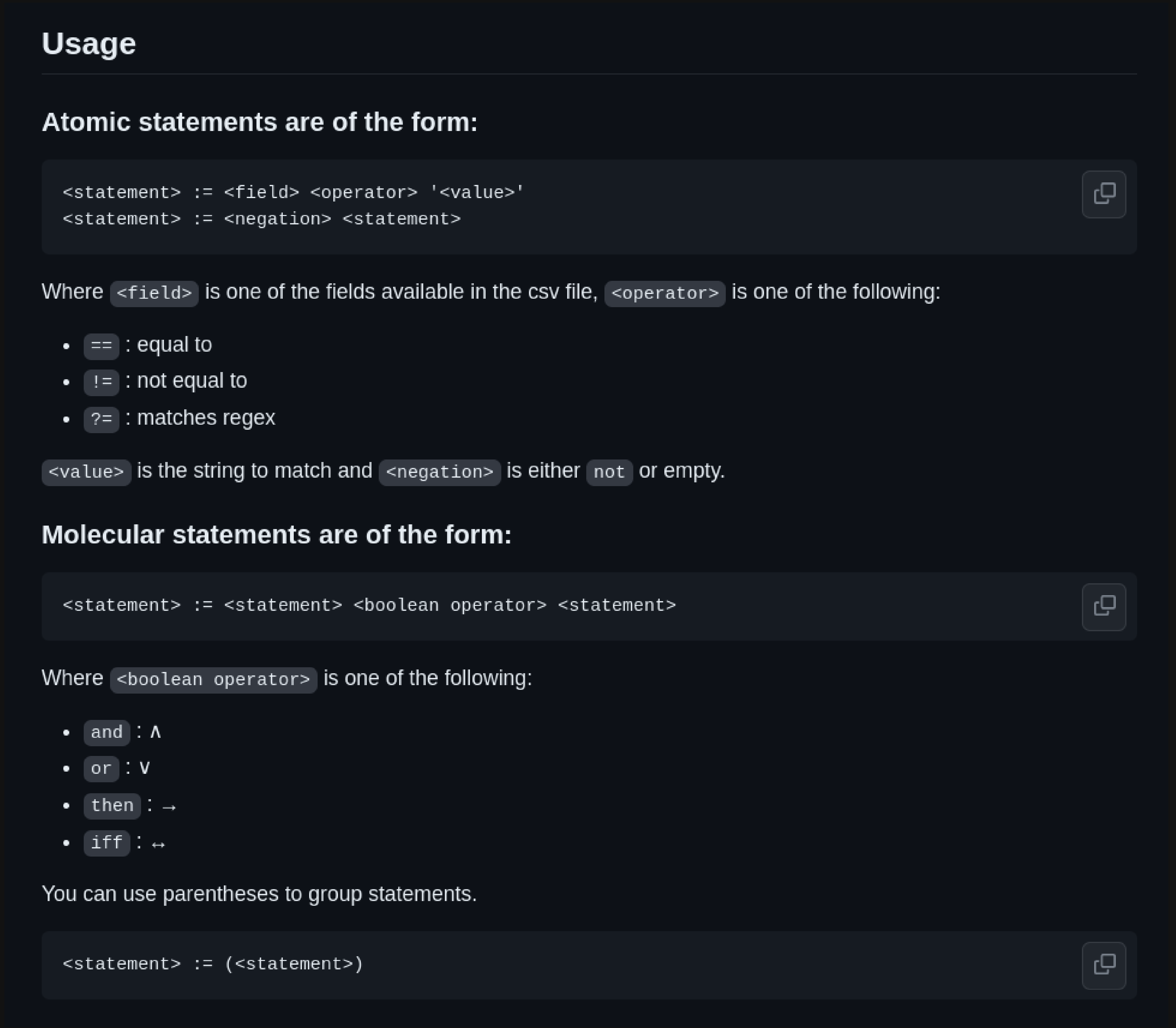
Languages

● Python 100.0%

```
dP                 dP
88                 88
88d888b. .d8888b. 88
88'  `88 88'  `88 88
88.  .88 88.  .88 88
88Y8888' `8888P88 dP
               88
               dP
```

boolean query language

GRAMÁTICA



Usage

Atomic statements are of the form:

```
<statement> := <field> <operator> '<value>'
<statement> := <negation> <statement>
```

Where `<field>` is one of the fields available in the csv file, `<operator>` is one of the following:

- `==` : equal to
- `!=` : not equal to
- `?=` : matches regex

`<value>` is the string to match and `<negation>` is either `not` or empty.

Molecular statements are of the form:

```
<statement> := <statement> <boolean operator> <statement>
```

Where `<boolean operator>` is one of the following:

- `and` : $\wedge$
- `or` : $\vee$
- `then` : $\rightarrow$
- `iff` : $\leftrightarrow$

You can use parentheses to group statements.

```
<statement> := (<statement>)
```

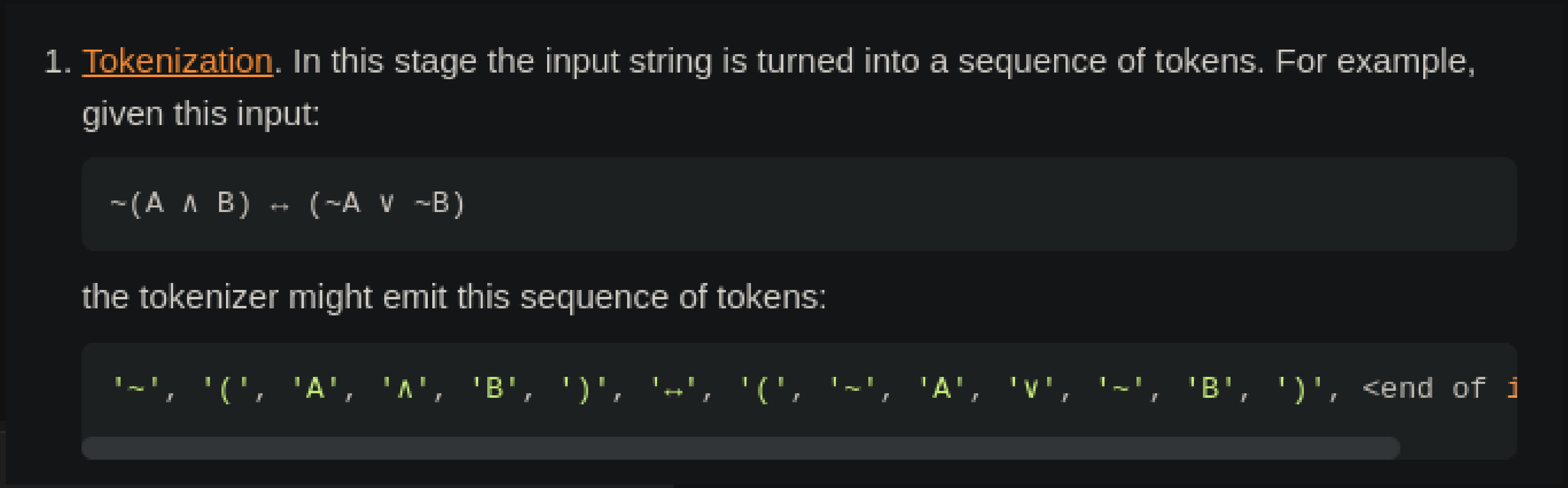
IMPLEMENTACIÓN

ALIASES

```python
def replace_logical_operators(input_string):
    pattern = r"(?<!['])\b(and|or|then|iff|not)\b(?!')(?=(?:[^']*'[^']*')*[^']*$)"
    operator_mapping = {'and': '∧', 'or': '∨', 'then': '→', 'iff': '↔', 'not': '~'}
    result = re.sub(pattern, lambda match: operator_mapping.get(match.group(0), match.group(0)), input_string)
    return result

def escape_parentheses_inside_quotes(input_string):
    pattern = r"([^']+)\((?=[^']*' |$)" # Opening parentheses
    result = re.sub(pattern, r"\1\(", input_string + " ")
    pattern = r"(\)([^')]*?)(?=[^']*'$|[^']*' ))" # Closing parentheses
    result = re.sub(pattern, r"\\\1", result[:-1])
    return result

def split_statement(input_string):
    # Split input string in atomic propositions
    pattern = r"(?<![\\])[\(\)]|∧|∨|→|↔|~" # Parentheses and logical operators
    propositions = re.split(pattern, input_string)
    propositions = [x.strip() for x in propositions if x.strip()] # Trim whitespace
    propositions = list(dict.fromkeys(propositions)) # Remove duplicates
    # Unscape parentheses
    propositions = [x.replace("\(", "(") for x in propositions]
    propositions = [x.replace("\)", ")") for x in propositions]
    return propositions

def set_boolean_aliases(original_input, propositions):
    # Associate each proposition with a boolean variable
    proposition_mapping = {}
    for i, proposition in enumerate(propositions):
        proposition_mapping[list(ascii_lowercase)[i]] = proposition
    for variable, proposition in proposition_mapping.items():
        original_input = original_input.replace(proposition, variable)
    original_input = replace_logical_operators(original_input)
    return (original_input, proposition_mapping)
```

```
>>> nombre ?= 'jack' and not (codigo ?= '^21' or escuela == 'soft')

a ∧ ~ (b ∨ c)

a: nombre ?= 'jack'
b: codigo ?= '^21'
c: escuela == 'soft'
```

TOKENIZER

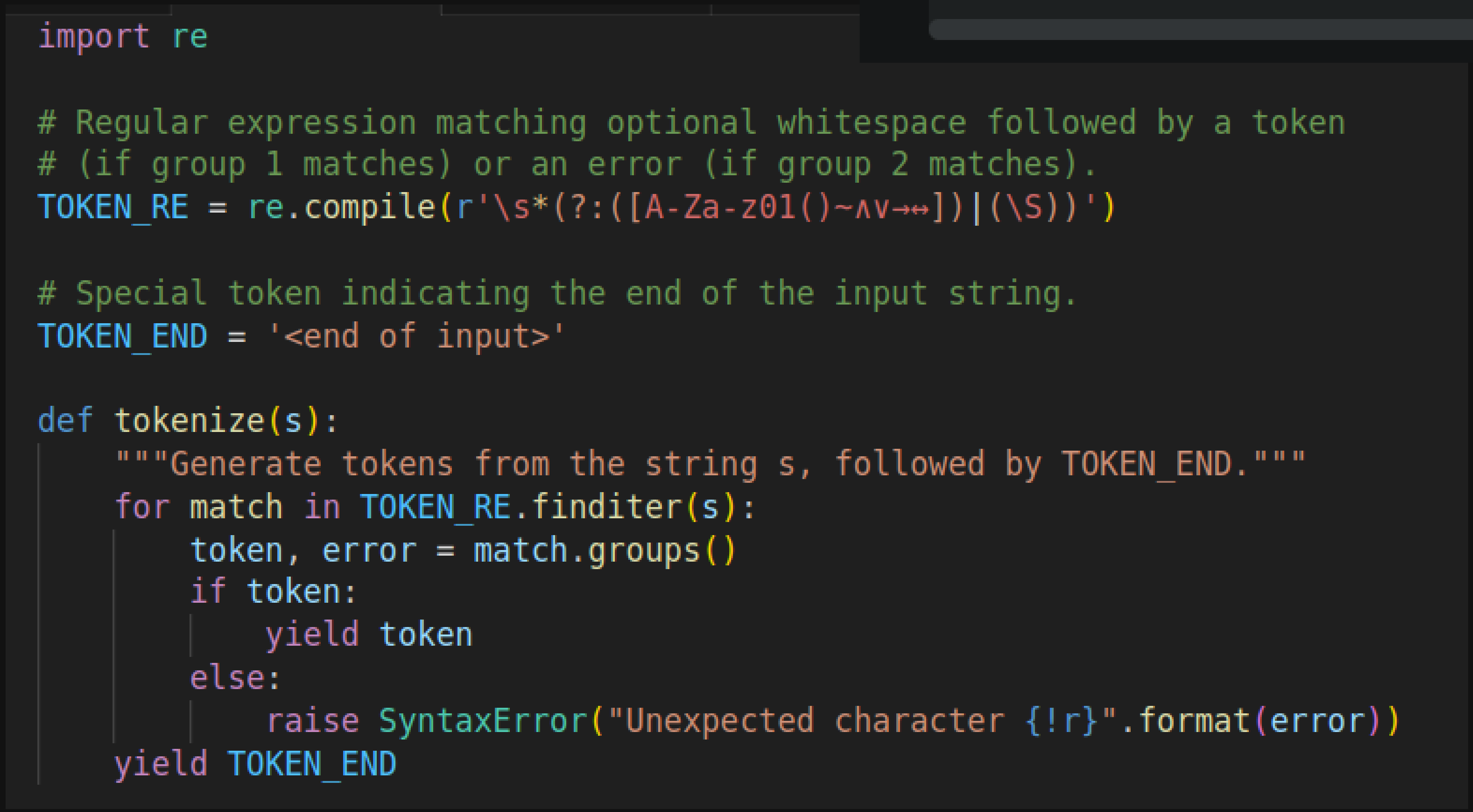


1. Tokenization. In this stage the input string is turned into a sequence of tokens. For example, given this input:

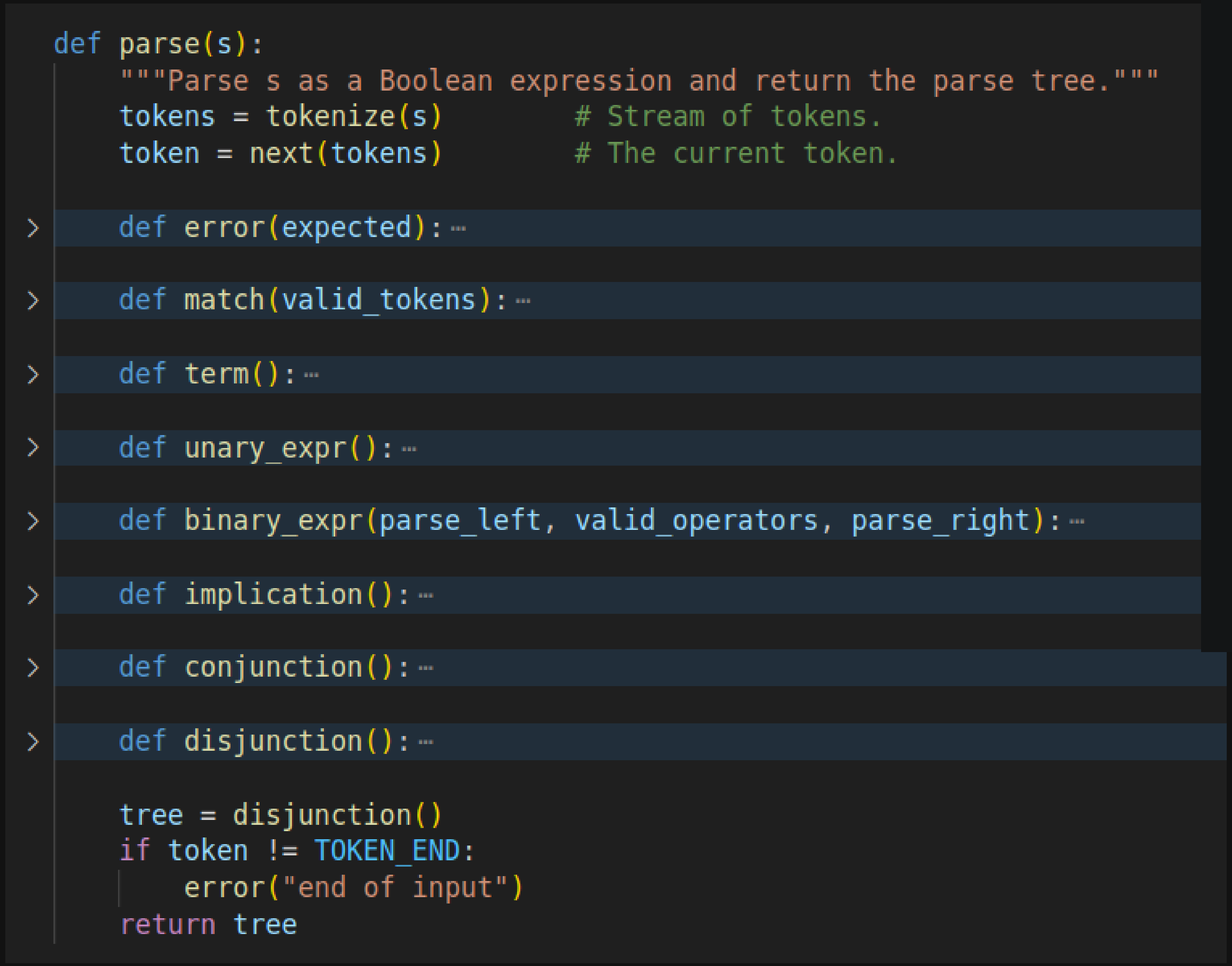
```
~(A ∧ B) ↔ (~A ∨ ~B)
```

the tokenizer might emit this sequence of tokens:

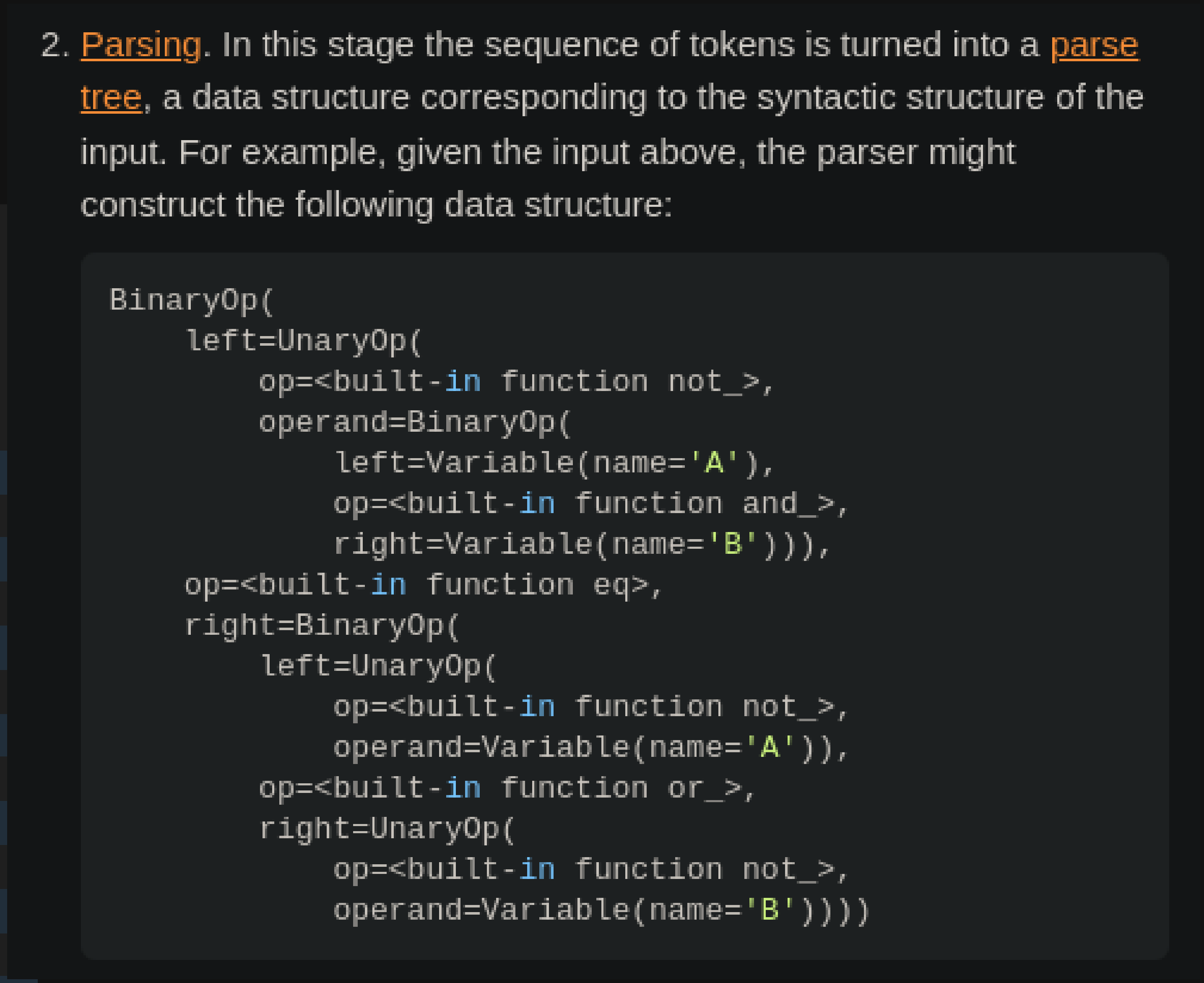
```
'~', '(', 'A', '∧', 'B', ')', '↔', '(', '~', 'A', '∨', '~', 'B', ')', <end of i
```



```python
import re

# Regular expression matching optional whitespace followed by a token
# (if group 1 matches) or an error (if group 2 matches).
TOKEN_RE = re.compile(r'\s*(?:([A-Za-z01()~∧∨→↔])|(\S))')

# Special token indicating the end of the input string.
TOKEN_END = '<end of input>'

def tokenize(s):
    """Generate tokens from the string s, followed by TOKEN_END."""
    for match in TOKEN_RE.finditer(s):
        token, error = match.groups()
        if token:
            yield token
        else:
            raise SyntaxError("Unexpected character {!r}".format(error))
    yield TOKEN_END
```



```python
def parse(s):
    """Parse s as a Boolean expression and return the parse tree."""
    tokens = tokenize(s)        # Stream of tokens.
    token = next(tokens)        # The current token.

    def error(expected): ⋯

    def match(valid_tokens): ⋯

    def term(): ⋯

    def unary_expr(): ⋯

    def binary_expr(parse_left, valid_operators, parse_right): ⋯

    def implication(): ⋯

    def conjunction(): ⋯

    def disjunction(): ⋯

    tree = disjunction()
    if token != TOKEN_END:
        error("end of input")
    return tree
```

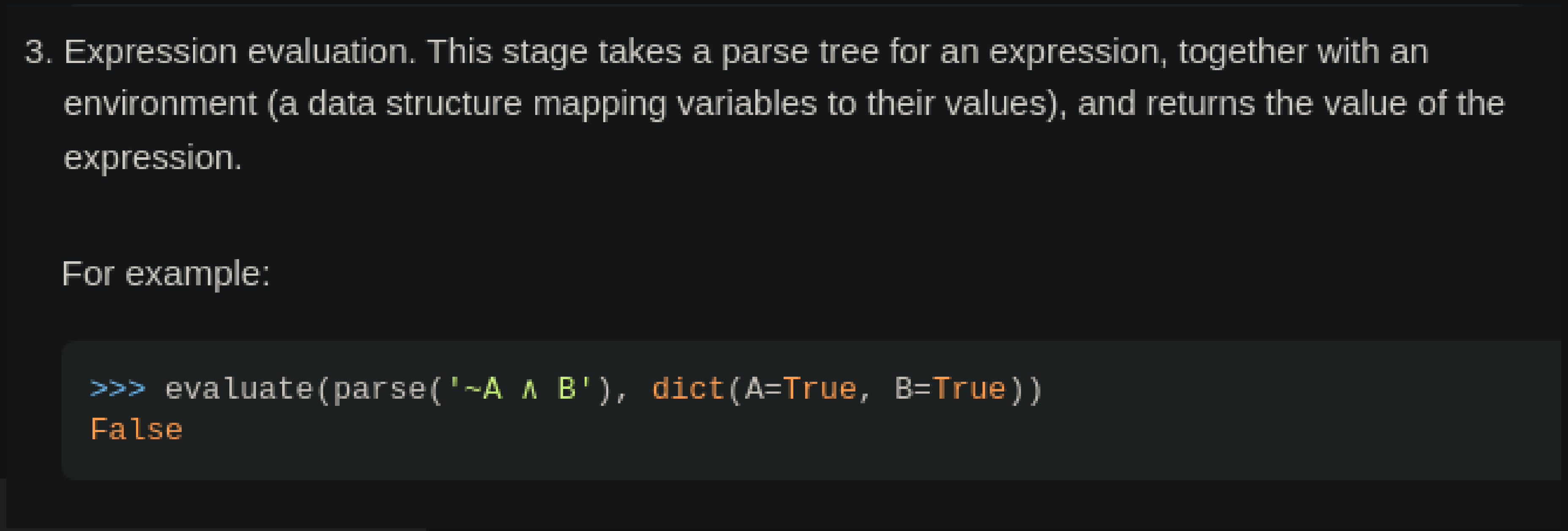


2. Parsing. In this stage the sequence of tokens is turned into a parse tree, a data structure corresponding to the syntactic structure of the input. For example, given the input above, the parser might construct the following data structure:

```
BinaryOp(
    left=UnaryOp(
        op=<built-in function not_>,
        operand=BinaryOp(
            left=Variable(name='A'),
            op=<built-in function and_>,
            right=Variable(name='B'))),
    op=<built-in function eq>,
    right=BinaryOp(
        left=UnaryOp(
            op=<built-in function not_>,
            operand=Variable(name='A')),
        op=<built-in function or_>,
        right=UnaryOp(
            op=<built-in function not_>,
            operand=Variable(name='B'))))
```

PARSER

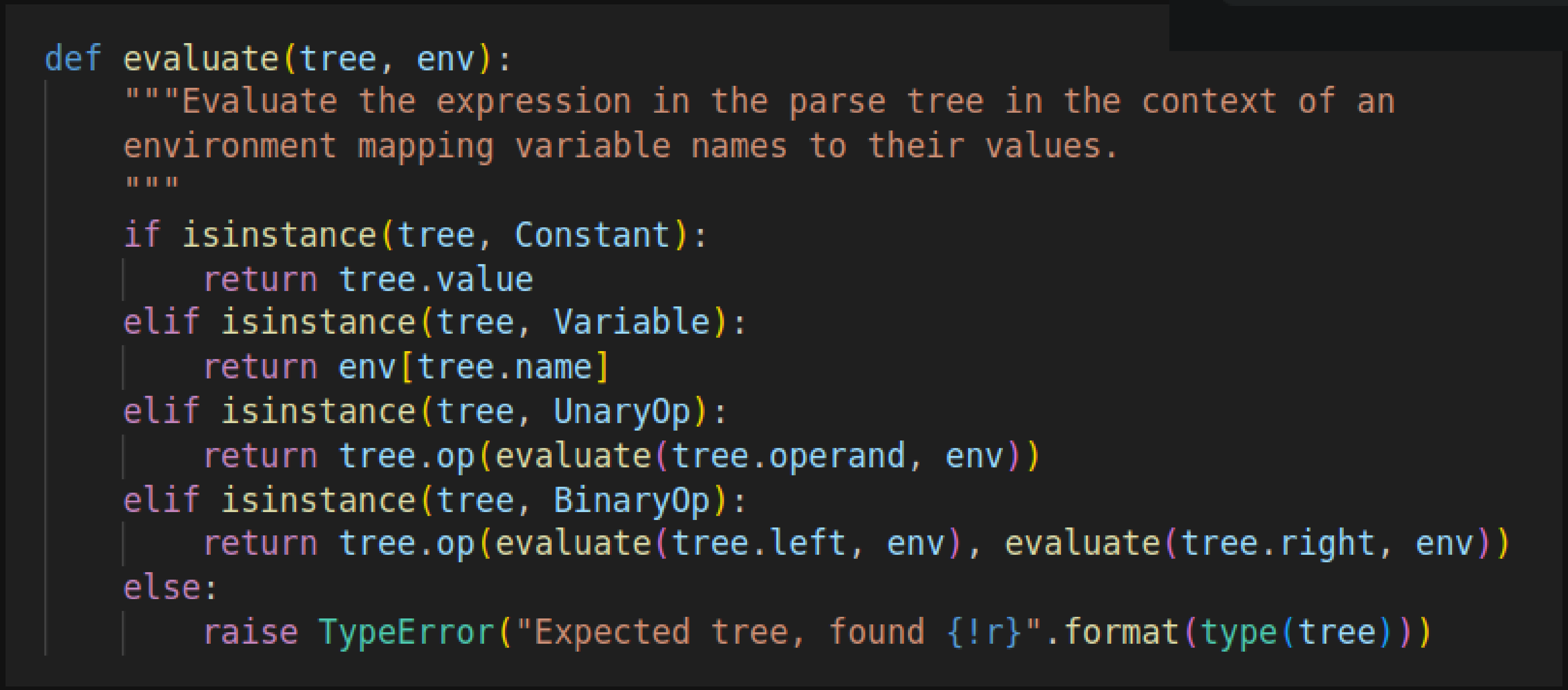
EVALUADOR



3. Expression evaluation. This stage takes a parse tree for an expression, together with an environment (a data structure mapping variables to their values), and returns the value of the expression.

For example:

```
>>> evaluate(parse('~A ∧ B'), dict(A=True, B=True))
False
```

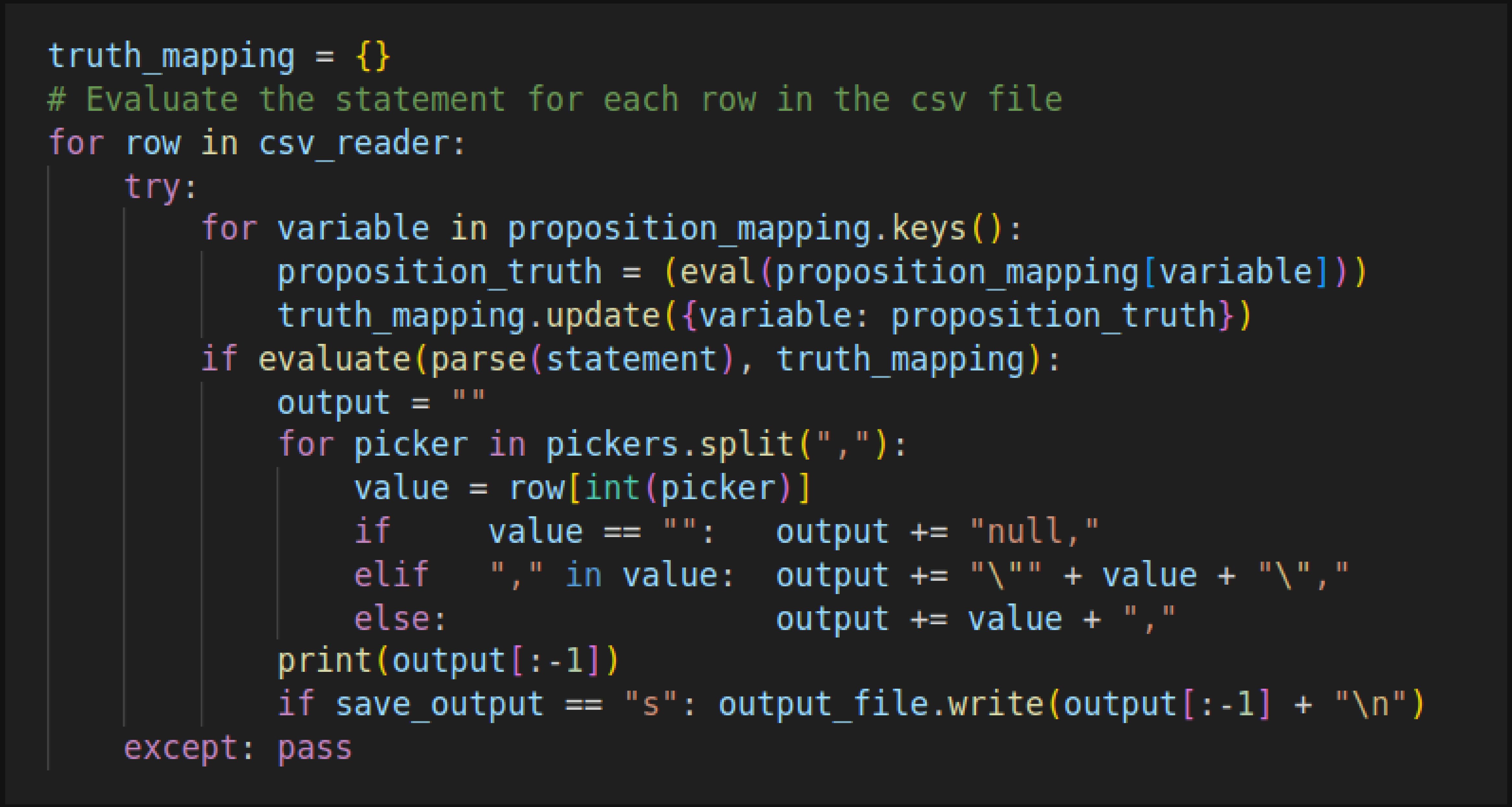


```python
def evaluate(tree, env):
    """Evaluate the expression in the parse tree in the context of an
    environment mapping variable names to their values.
    """
    if isinstance(tree, Constant):
        return tree.value
    elif isinstance(tree, Variable):
        return env[tree.name]
    elif isinstance(tree, UnaryOp):
        return tree.op(evaluate(tree.operand, env))
    elif isinstance(tree, BinaryOp):
        return tree.op(evaluate(tree.left, env), evaluate(tree.right, env))
    else:
        raise TypeError("Expected tree, found {!r}".format(type(tree)))
```

```python
# Format input propositions as python code
l_equal_pattern = r"(\S+)\s*(?= ==| !=)" # in "A == 'B'" selects A
l_regex_pattern = r"(\S+)\s*(?= \?=)" # in "A ?= 'B'" selects A
r_pattern = r"(?<=(== |\?= )')[\s\S]+?(?=')" # in "A == 'B'" or "A ?= 'B'" selects B

code_propositions = propositions[:]
for i in range(len(code_propositions)):
    for j in range(len(header)):
        try:
            if re.search(l_equal_pattern, code_propositions[i]).group(0) == header[j]:
                code_propositions[i] = code_propositions[i].replace(header[j], field_indexes[header[j]])
        except: pass
        try:
            if re.search(l_regex_pattern, code_propositions[i]).group(0) == header[j]:
                field_index = field_indexes[header[j]]
                r_val = str(re.search(r_pattern, code_propositions[i]).group(0))
                code_propositions[i] = "re.search(r\"" + r_val + "\", " + field_index + ", re.IGNORECASE) is not None"
        except: pass

for variable, proposition in proposition_mapping.items():
    proposition_mapping[variable] = code_propositions[propositions.index(proposition)]
```

FORMATO

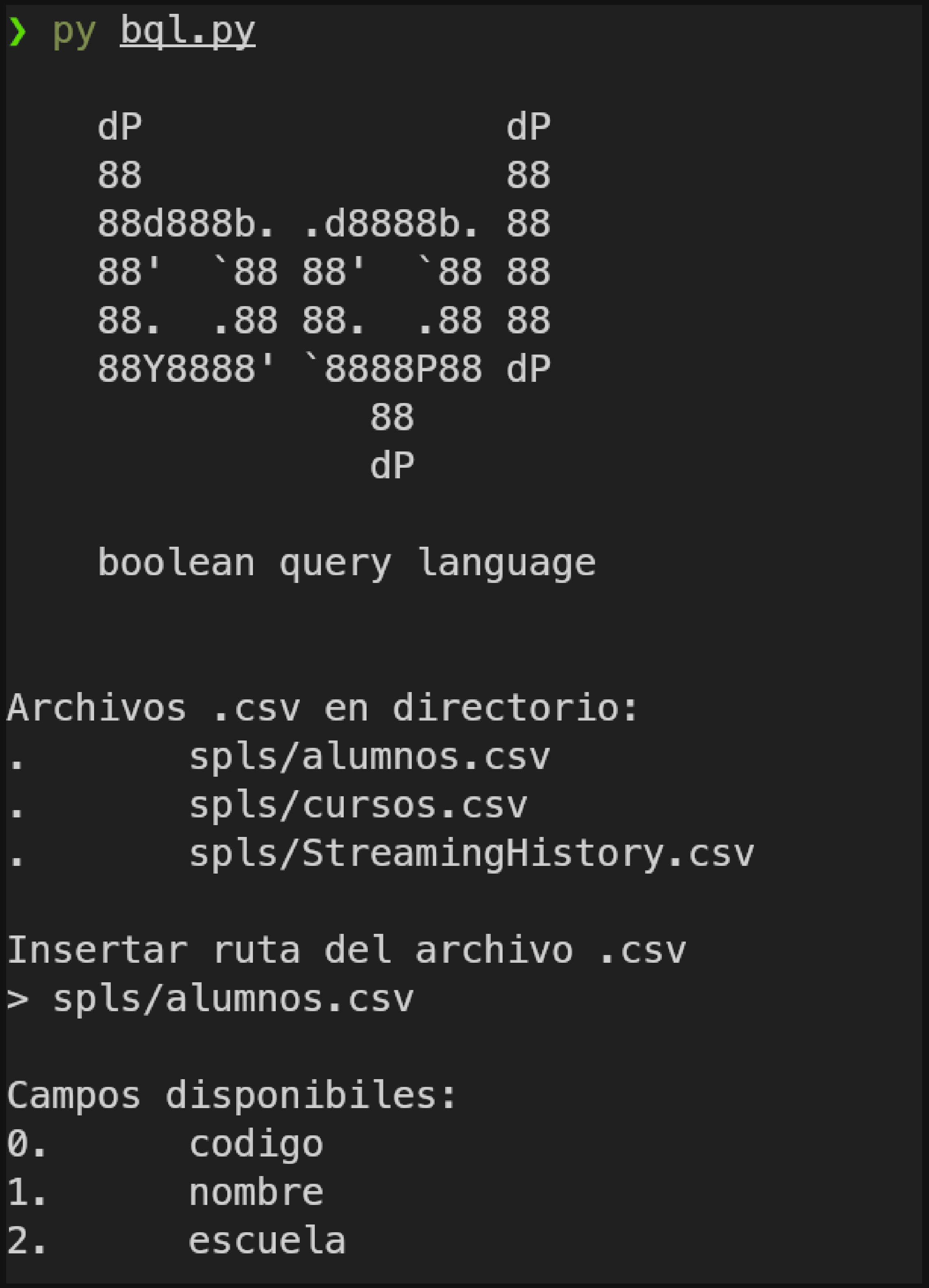
```
>>> a: nombre ?= 'jack'
>>> b: codigo ?= '^21'
>>> c: escuela == 'soft'

a: re.search(r"jack", row[1], re.IGNORECASE) is not None
b: re.search(r"^21", row[0], re.IGNORECASE) is not None
c: row[2] == 'soft'
```



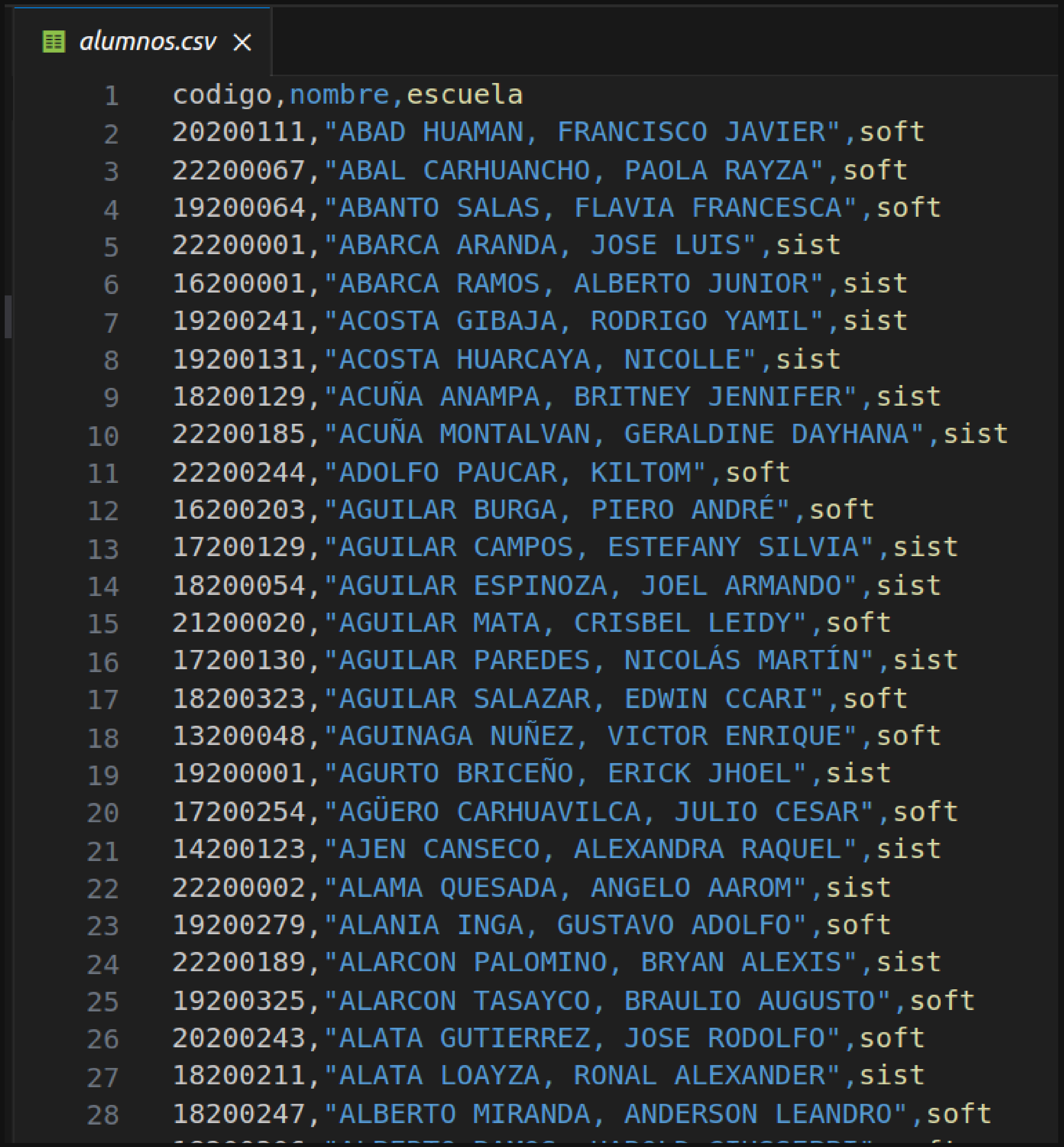
```python
truth_mapping = {}
# Evaluate the statement for each row in the csv file
for row in csv_reader:
    try:
        for variable in proposition_mapping.keys():
            proposition_truth = (eval(proposition_mapping[variable]))
            truth_mapping.update({variable: proposition_truth})
        if evaluate(parse(statement), truth_mapping):
            output = ""
            for picker in pickers.split(","):
                value = row[int(picker)]
                if     value == "":   output += "null,"
                elif   "," in value:  output += "\"" + value + "\","
                else:                 output += value + ","
            print(output[:-1])
            if save_output == "s": output_file.write(output[:-1] + "\n")
    except: pass
```

ITERAR Y MOSTRAR

EJEMPLO



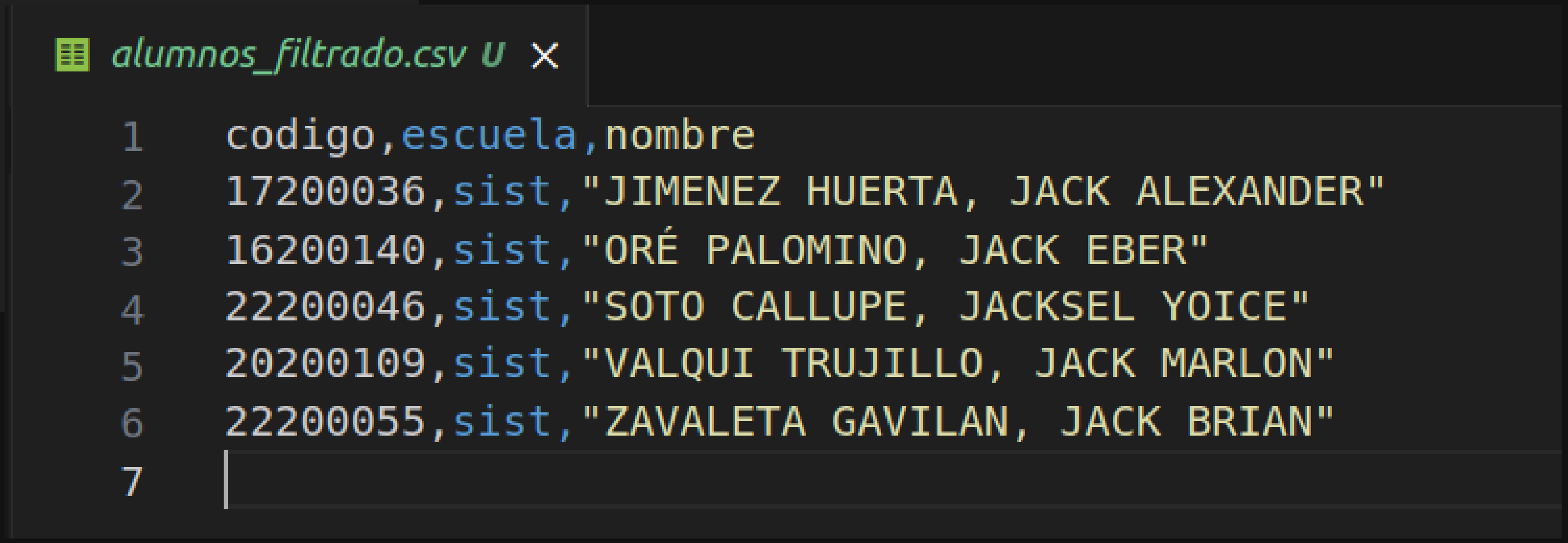
```
❯ py bql.py

    dP                  dP
    88                  88
    88d888b. .d8888b. 88
    88'  `88 88'  `88 88
    88.  .88 88.  .88 88
    88Y8888' `8888P88 dP
                  88
                  dP

    boolean query language


Archivos .csv en directorio:
.       spls/alumnos.csv
.       spls/cursos.csv
.       spls/StreamingHistory.csv

Insertar ruta del archivo .csv
> spls/alumnos.csv

Campos disponibiles:
0.      codigo
1.      nombre
2.      escuela
```



alumnos.csv

```
codigo,nombre,escuela
20200111,"ABAD HUAMAN, FRANCISCO JAVIER",soft
22200067,"ABAL CARHUANCHO, PAOLA RAYZA",soft
19200064,"ABANTO SALAS, FLAVIA FRANCESCA",soft
22200001,"ABARCA ARANDA, JOSE LUIS",sist
16200001,"ABARCA RAMOS, ALBERTO JUNIOR",sist
19200241,"ACOSTA GIBAJA, RODRIGO YAMIL",sist
19200131,"ACOSTA HUARCAYA, NICOLLE",sist
18200129,"ACUÑA ANAMPA, BRITNEY JENNIFER",sist
22200185,"ACUÑA MONTALVAN, GERALDINE DAYHANA",sist
22200244,"ADOLFO PAUCAR, KILTOM",soft
16200203,"AGUILAR BURGA, PIERO ANDRÉ",soft
17200129,"AGUILAR CAMPOS, ESTEFANY SILVIA",sist
18200054,"AGUILAR ESPINOZA, JOEL ARMANDO",sist
21200020,"AGUILAR MATA, CRISBEL LEIDY",soft
17200130,"AGUILAR PAREDES, NICOLÁS MARTÍN",sist
18200323,"AGUILAR SALAZAR, EDWIN CCARI",soft
13200048,"AGUINAGA NUÑEZ, VICTOR ENRIQUE",soft
19200001,"AGURTO BRICEÑO, ERICK JHOEL",sist
17200254,"AGÜERO CARHUAVILCA, JULIO CESAR",soft
14200123,"AJEN CANSECO, ALEXANDRA RAQUEL",sist
22200002,"ALAMA QUESADA, ANGELO AAROM",sist
19200279,"ALANIA INGA, GUSTAVO ADOLFO",soft
22200189,"ALARCON PALOMINO, BRYAN ALEXIS",sist
19200325,"ALARCON TASAYCO, BRAULIO AUGUSTO",soft
20200243,"ALATA GUTIERREZ, JOSE RODOLFO",soft
18200211,"ALATA LOAYZA, RONAL ALEXANDER",sist
18200247,"ALBERTO MIRANDA, ANDERSON LEANDRO",soft
```

```
Insertar formato de salida (ej. 0, 3, 4)
> 0, 2, 1

Insertar filtros (ej. nombre ?= 'regex' and codigo != 'valor')
> nombre ?= 'jack' and not (codigo ?= '^21' or escuela == 'soft')

¿Guardar salida en un archivo? (s/n)
> s

Insertar nombre del archivo de salida
> alumnos_filtrado.csv

codigo,escuela,nombre
17200036,sist,"JIMENEZ HUERTA, JACK ALEXANDER"
16200140,sist,"ORÉ PALOMINO, JACK EBER"
22200046,sist,"SOTO CALLUPE, JACKSEL YOICE"
20200109,sist,"VALQUI TRUJILLO, JACK MARLON"
22200055,sist,"ZAVALETA GAVILAN, JACK BRIAN"
```



```
codigo,escuela,nombre
17200036,sist,"JIMENEZ HUERTA, JACK ALEXANDER"
16200140,sist,"ORÉ PALOMINO, JACK EBER"
22200046,sist,"SOTO CALLUPE, JACKSEL YOICE"
20200109,sist,"VALQUI TRUJILLO, JACK MARLON"
22200055,sist,"ZAVALETA GAVILAN, JACK BRIAN"
```

¡GRACIAS!