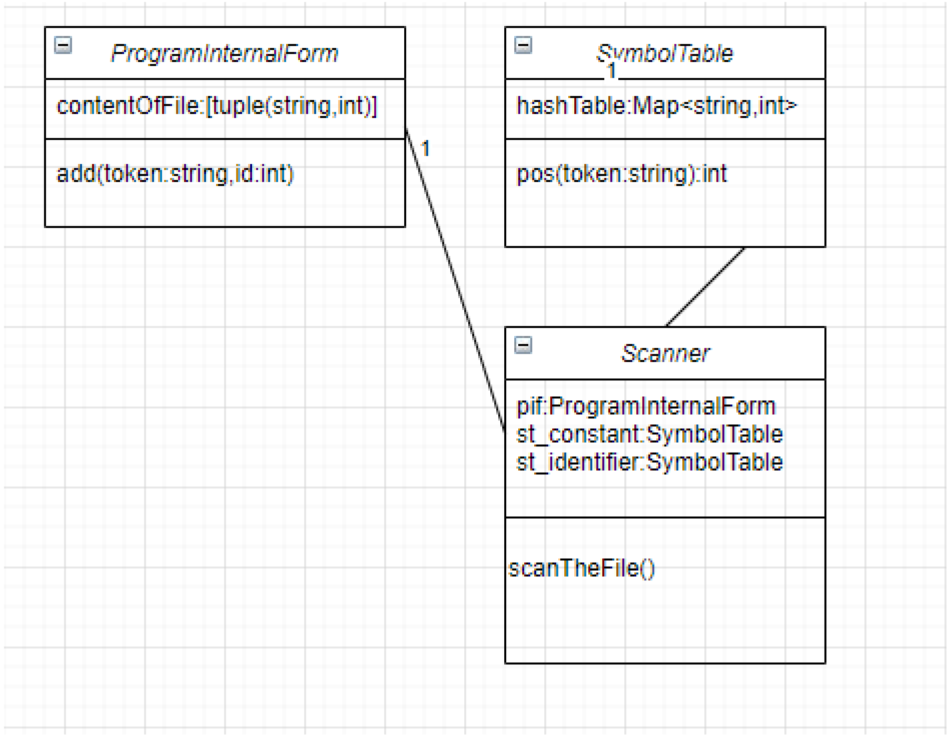
Documentation for the lab3:



ProgramInternalForm:

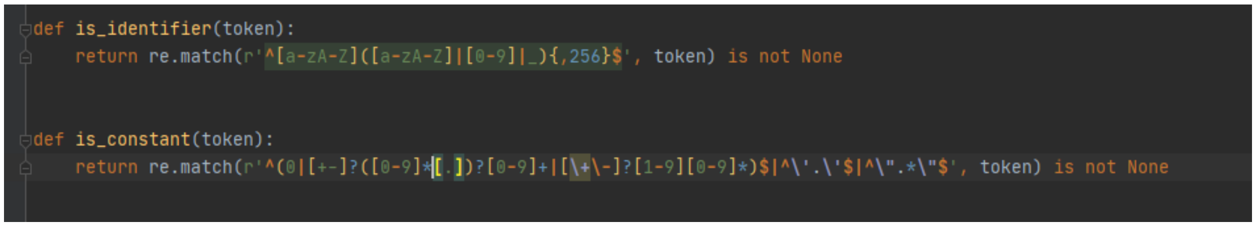
contentOfFile is a list of tuples containing the Program Internal Form, first element is the token and the second is an integer

‘add’ adds the token to the symbol table

The SymbolTable is a HashTable containing the with string int pairs for the token and the code in the symbol table

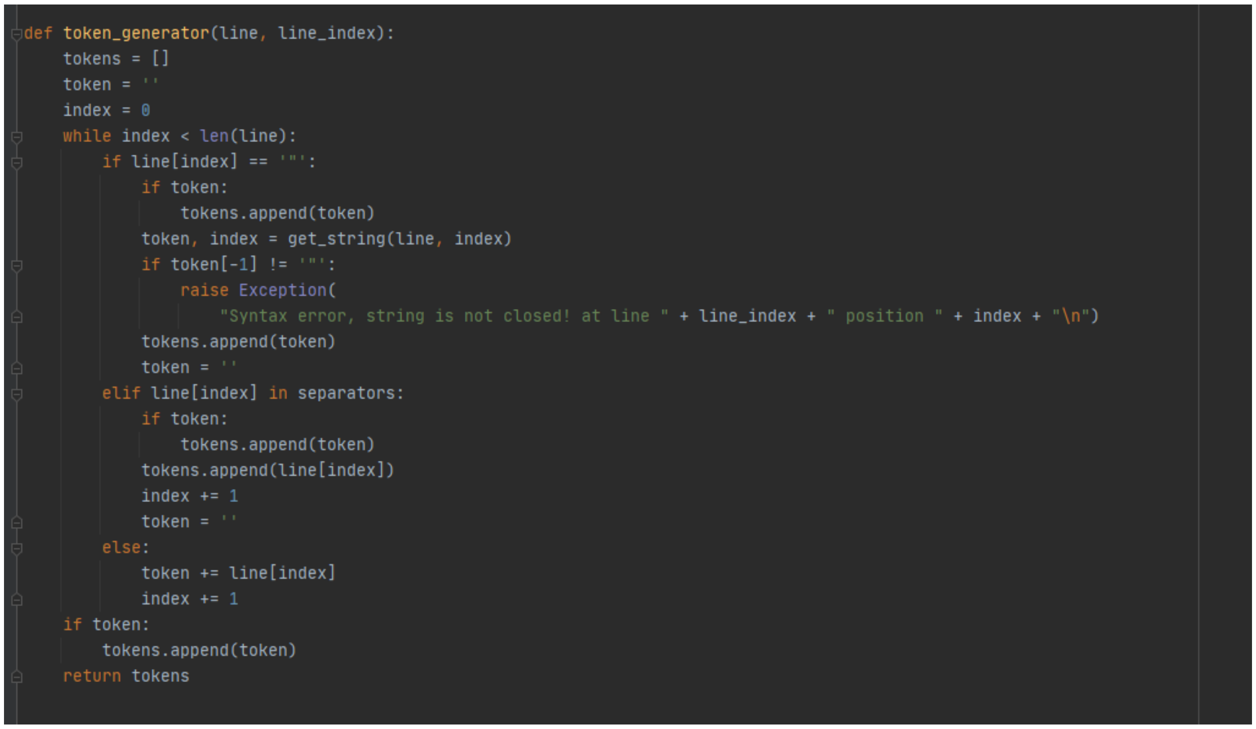
‘pos’ adds the token to the symbol table if it doesn’t exist otherwise it gets it’s position in the PIF

The Scanner has a Program Internal Form and to SymbolTables to handle the different constants and identifiers



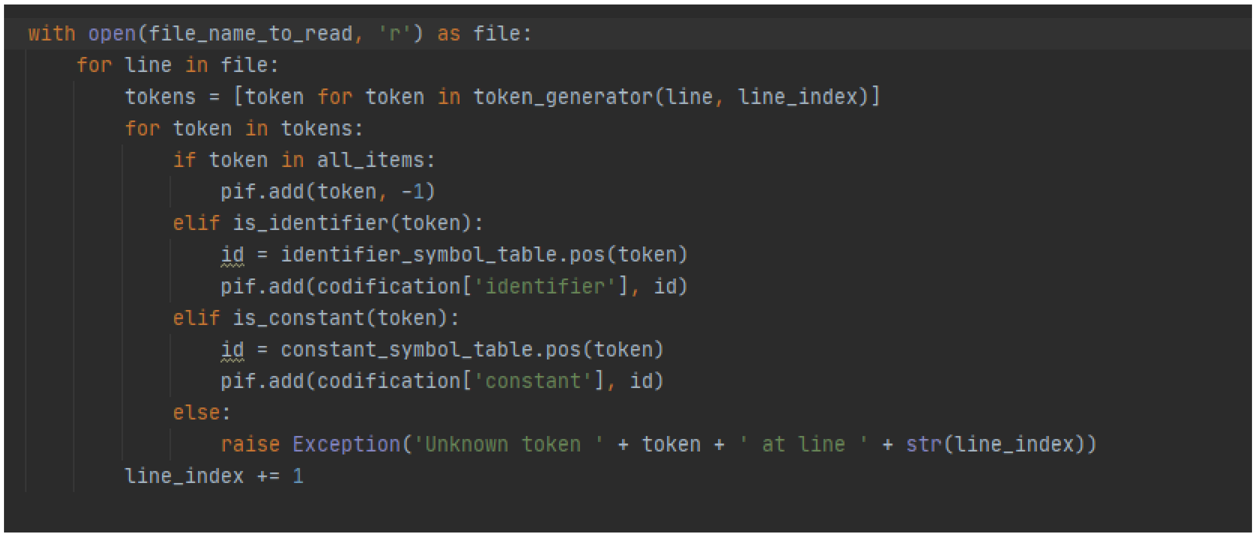
```
def is_identifier(token):
    return re.match(r'^[a-zA-Z]([a-zA-Z]|[0-9]|_){,256}$', token) is not None


def is_constant(token):
    return re.match(r'^(0|[+-]?([0-9]*[.])?[0-9]+|[\+\-]?[1-9][0-9]*)$|^\'.\'$|^\".*\"$', token) is not None
```

Regexes that handle the identification of the tokens



```python
def token_generator(line, line_index):
    tokens = []
    token = ''
    index = 0
    while index < len(line):
        if line[index] == '"':
            if token:
                tokens.append(token)
            token, index = get_string(line, index)
            if token[-1] != '"':
                raise Exception(
                    "Syntax error, string is not closed! at line " + line_index + " position " + index + "\n")
            tokens.append(token)
            token = ''
        elif line[index] in separators:
            if token:
                tokens.append(token)
            tokens.append(line[index])
            index += 1
            token = ''
        else:
            token += line[index]
            index += 1
    if token:
        tokens.append(token)
    return tokens
```

The algorithm that generates the tokens line for a line, it goes character by character and if it finds a string initial quote is starts a search for a string if it doesn't find it signals the error that happened on the line, otherwise if it is a separator it adds the currently created token to the list then adds the separator to the list of tokens, if it is neither of these it continues to construct the token, in the end the last token is added



```python
with open(file_name_to_read, 'r') as file:
    for line in file:
        tokens = [token for token in token_generator(line, line_index)]
        for token in tokens:
            if token in all_items:
                pif.add(token, -1)
            elif is_identifier(token):
                id = identifier_symbol_table.pos(token)
                pif.add(codification['identifier'], id)
            elif is_constant(token):
                id = constant_symbol_table.pos(token)
                pif.add(codification['constant'], id)
            else:
                raise Exception('Unknown token ' + token + ' at line ' + str(line_index))
        line_index += 1
```

Categorization similarly to the course algorithm, if it all_items refers to separators, reserwed words and operators, is_identifer uses the regex to check if it is an identifier, is_constant check if it is a 0 digit or a real number or an integer or a single character or a string, if it is neither of these then it is an error

```
'^[a-zA-Z]([a-zA-Z]|[0-9]|_){,255}$'
```

This regex is used to detect the identifiers, it says that at the start it has to be a character then, it’s a letter, or digit, or then it says that following the first character it can be and ‘_’ or digit or lower or uppercase letter, the string can be maximum 256 characters, which is represented by the 255 at the end, which refers to the part between ( )

```
^(0|[+-]?([0-9]*[.])?[0-9]+|[\+\-]?[1-9][0-9]*)$|^\'.\'$|^\".*\"$
```

The regex to match the constants, first the 0 number itself, the following represents real number as with a + or – then it can be a real number with a mantissa or not, following that we can have an integer with an optional + or – at the start followed by at least one digit, the next is the regex for a single character, then the regex for a string, where it needs to have ‘ at the start and the end and “ at the start and the end.