CS417 Lab #16

Getting Started

Begin the lab by downloading this starting file:

- `lexer.py`

Lexical Analysis

In this lab, you will build a lexical analyzer. Lexical analysis is the first stage in a compiler or interpreter. It takes an expression in a program, such as

```
   cost = total*1.5
```

and does two things:

First, it breaks the expression into tokens:

```
   ['cost', '=', 'total', '*', '1.5']
```

Second, it identifies the meaning of each token:

```
       ('cost' , 'variable')
       ('='    , 'operator')
       ('total', 'variable')
       ('*'    , 'operator')
       ('1.5'  , 'number'  )
```

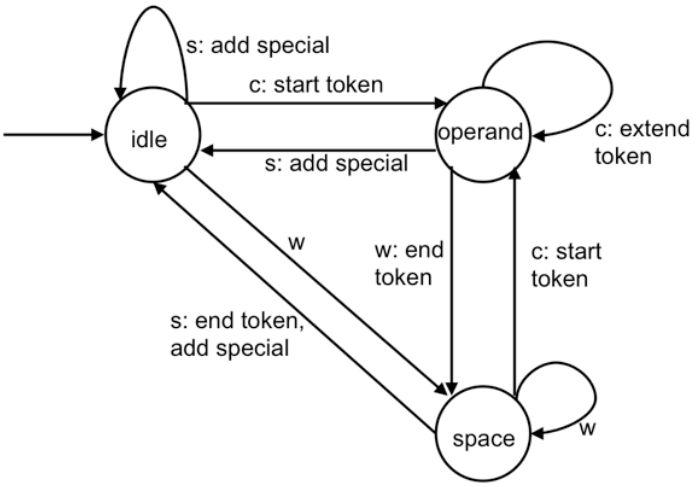
In this lab, I have broken this task up into two functions:

- `tokenize` breaks the expresion into tokens
- `lexer` identifies each token's type

Your Tasks

1. Implement `tokenize()`

   This function does work similar to the `split` function, but is smarter. It implements the following state machine:



The machine handles three kinds of characters:

- c : an ordinary character, which is part of an operand
- s : a special, usually an operator, in `"+-*/()="`
- w : whitespace, usually spaces, in `" \t"`

Each transition has an associated action. Examples:

- the IDLE → OPERAND transition should start a token.
- the OPERAND → OPERAND transition should extend the token (`token += c`)
- the OPERAND → SPACE transition should save the token (append it to `tokens`).

Open `lexer.py` and look at the `tokenize` function. It expects an expression, a set of special characters, and a set of whitespace characters.

Notice the `for` loop in `tokenize`. I have already handled all transitions out of the IDLE state.

You should implement the other transitions.

---

2. Consider this expression: `"abc + def"`. When the state machine finishes, you will be in the OPERAND state. However, you need an extra transition to actually *save* the operand, either

   - OPERAND → SPACE or
   - OPERAND → IDLE.

   So, there may be an operand waiting to be saved, if you finish in the OPERAND state, *after* the `for` loop (un-indent!):

```
if state == OPERAND:
    tokens.append(operand)
```

---

3. Implement `lexer(tokens)`.

   We want to identify 4 kinds of tokens:

   - `'operator'`

- `'number'`
- `'variable'`
- `'unknown'`

Operators are easy: if the token occurs in `operators`, set `lex_type = 'operator'`:

```
if token in operators:
    lex_type = 'operator'
```

else, it's a number, or a variable. Numbers are a bit harder, but python can help. Try to convert the token into a `float`. If you succeed, it's a number. If not, it might be a variable:

```
try:
    x = float(token)
    lex_type = 'number'
except ValueError:
    (maybe it's a variable?)
```

---

4. Is it a valid variable name?

   Finally, check if the token follows the rules for variable names. If so, its type is `'variable'`. If not, it is `'unknown'`.

   The rules for a python variable name are:

   - it must start with a letter or an underscore.
   - the remaining characters may be letters, digits, or underscores.

   Hint: look at the python documentation: `docs.python.org/3/library/string.html`. You may find these constants useful:

   - `string.ascii_letters`
   - `string.digits`

   The first character must be `in string.ascii_letters + '_'`, and

   The other characters must be `in string.ascii_letters + '_' + string.digits`.

Testing

If your program works correctly, you should get this output:

```
Tokenize "hello, how are,you?" :
   ['hello', 'how', 'are', 'you?']
Tokenize "cost= total + (7.0 / 100)* total" :
   ['cost', '=', 'total', '+', '(', '7.0', '/', '100', ')', '*', 'total']

Analyze ['cost','=','total','+','(','7.0','/','100',')','*','total']:
    cost variable
       = operator
   total variable
       + operator
       ( operator
     7.0 number
       / operator
     100 number
       ) operator
       * operator
   total variable
```

Turning in your work

To submit your work, go to mycourses.unh.edu, find cs417, find the lab, and upload `lexer.py`. Submit whatever you have completed, at the end of the lab session. You can submit again until midnight, with no lateness penalty.