Parse Assignment Statements

Add the following rule to the recursive descent parser:

<statement> → <id> = <exp>

The new start symbol is <statement>

Define a new function named **statement()** to parse this rule. On a successful parse, **statement()** should call a function

assign(id, value)

The assign function is part of the parser back end, value is what the call to exp() returns. You could do **assign(id, exp())**.

The parser will now keep a symbol table. This will be a Python dictionary named **symbol_table** where the key values are the identifiers, and the values are the expression values. The assign function should make an entry in the symbol table.

We will also modify the atomic() function. At present it assumes the only atomic value (a leaf node in the parse tree) is a numeric literal. Now we will add the possibility it could be a variable. In psuedo-code:

```
atomic(token)
     try to convert token to a float.
          On success, return the float
     on ValueError exception:
          look for token in the symbol table.
               if found
                    return the value
               else
                    error: “float literal or variable expected”
```