1. **Download the** zipped file with all of the starting code

2. **Implement the following subset the Python grammar:**

    1. `if`, `elif`, `else`
    2. `while`
    3. `print` (as a `small_stmt`)
    4. integers, floats, "strings", 'strings', `True`, `False`
    5. <, <=, >, >=, ==, !=
    6. +, -, *, /, ==, !=
    7. = (assignment)
    8. and, or, not

3. **Decide the early Bison structure**

    1. In `ourPython.y`, please defined `%token`s for
        - `NEWLINE` (already done for you)
        - semicolons, comma
        - begin parentheses, end parentheses
        - `if`, `elif`, `else`
        - `while`
        - `print` (as a `small_stmt`)
        - integers, floats, "strings", 'strings', `True`, `False`
        - <, <=, >, >=, ==, !=
        - +, -, *, /, ==, !=
        - = (assignment)
        - and, or, not

        You may choose to define distinct tokens for similar operators. Or, you may choose to group similar operators together. For example, the Python grammar groups the comparison operators (e.g. <, <=, >, >=, ==, !=) together as `COMP`.

    2. You need to keep track of the following data for various tokens:
        - Perhaps `operator_ty`: to distinguish among operators (e.g. `OR_OP`, `PLUS_OP`, etc.)
        - Perhaps `Object*`: to represent integers, floats, strings, `True` and `False`)
        - `Expression*`: to represent `ConstantExpression`, `VariableExpression`, `UnaryExpression`, `BinaryExpression` and `AssignmentExpression`)
        - `Statement*`: to represent `ExpressionStatement`, `PrintStatement`, `IfThenElseStatement`, `WhileStatement`, `BlockStatement`.

Therefore, use `%union` (recommended) or `YYSTYPE`

3. Please use `single_input` (already given to you) as the starting non-terminal. Please also decide which non-terminals to use, and what `%type` they should have.

4. **Write the Flex tokenizing rules**

   Please keep my rules for `#` style comments, newlines, spaces and tabs.

   Please fill in your rules where the comment tells you to.

   You may also define regular expressions to help you. Where or not you do or don&t, please remove the line

   ```
   // Erase this line and perhaps put patterns here
   ```

5. **Write the Bison parse rules**

   Please go to https://docs.python.org/3/reference/grammar.html and encode the subset of the grammar that you will need. We will make a number of simplifications:

   - Only implement the operators specified above
   - Implement `print` as a `small_stmt`
   - Our `print` can only have 0 or 1 operands
   - Do not worry about turning `x < y < z` into `x < y and y < z`

   Combine the `$1`'s, `$2`'s, `$3`'s, etc, to make `$$`.

   Class `Statement` has the following methods to help you:

   - `void appendElif(Expression* condPtr,Statement* thenPtr)` appends a new `IfThenElseStatement` at the end of a chain of them.
   - `void appendElse(Statement* elsePtr)` appends ending `else` code at the end of a chain of `IfThenElseStatement`s.
   - `void addStatement(Statement* statePtr)` appends `statePtr` at the end of a block of code.
     (See below.)

Sample output:

I gave you 6 python programs: 1.py, 2.py, 3.py, 3.py, 5.py and computeBMI.py.
Their output from your program should be the same as the output from ordinary Python. You may check this with:

```
$ python 5.py > expected.out
$ ./ourPython 5.py > test.out
$ diff expected.out test.out
```

If the files `expected.out` and `test.out` are identifical then the `diff` program will have no output.