CS314 HW 1: Imperative programming and Python

Spring 2019

Implement the following in Python. You should not use any libraries that do all the heavy lifting for you.

1. **matmult.py:** Matrix multiplication. Two matrices will be provided via stdin, first with the number of rows and columns, followed by the matrix values, separated by a single space:

   Example input (a 2×3 matrix and a 3×1 matrix):

   ```
   2 3
   1.2 2.3 3.4
   4.5 5.6 6.7
   3 1
   6.54
   5.43
   4.32
   ```

   If the two matrices can be multiplied, print the resulting matrix in the same format. If they cannot be multiplied, print "invalid input". You can assume the input will always be formatted correctly, and no matrix will have 0 rows or 0 columns.

2. **bst.py:** Implement a binary search tree class (without balancing). Commands will be given one per line via stdin – `i` for insert and `q` for query.

   For insert commands, you should not print any output.

   For query commands, you should report "not found" if the value was not found. If it was found, you should report "found: root" if it's the root element, or the sequence of left/right edges to follow to reach it otherwise, e.g., "found: r r l r".

   Example:

   ```
   i 1
   i 2
   ```

```
i 3
q 1    # prints "found: root"
q 2    # prints "found: r"
q 3    # prints "found: r r"
q 4    # prints "not found"
```

3. **rpn.py:** Write an interpreter for a reverse Polish notation (RPN) calculator. Each line of the input corresponds to a single operand or operator. Operands should be pushed onto a stack, and popped as needed when an operator is encountered (and the result pushed back on the stack). At each step, you should print the top-most element on the stack. If an operator requires more elements than are available on the stack, you should print "invalid operation" and ignore the operator. Operands will be non-negative integers. You should support these operators:

`+`	Add two numbers
`-`	Subtract two numbers
`*`	Multiply two numbers
`/`	Divide two numbers
`~`	Negate one number

   Example:

   ```
   23   # prints 23
   5    # prints 5
   +    # pops 23 and 5, pushes and prints 28
   ```

   Another example:

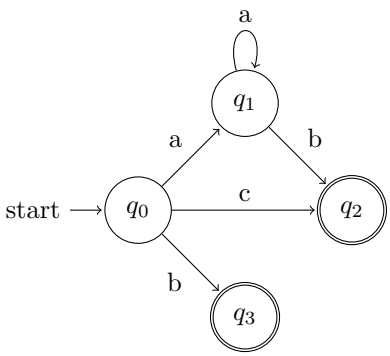
   ```
   1    # prints 1
   2    # prints 2
   +    # pops 1 and 2, pushes and prints 3
   5    # prints 5
   1    # prints 1
   -    # pops 5 and 1, pushes and prints 4
   +    # pops 3 and 4, pushes and prints 7
   ```

4. **dfa.py:** Write a program that reads a description of a deterministic finite automaton (DFA) and then classifies input strings as accepted or rejected by the DFA.

   DFAs are characterized by the following 5-tuple: $(Q, \Sigma, \delta, q_0, F)$, where Q denotes the set of states, Σ is the alphabet of possible input symbols, δ is the set of transition rules, q_0 is the start state, and F is the set of final (accepting) states.

Input to the program will be a DFA specification followed by a number of input strings. For each input string, you should print "accepted" or "rejected".

Example DFA:



Corresponding input, including input test strings:

```
states: q0 q1 q2 q3
symbols: a b c
begin_rules
q0 -> q1 on a
q1 -> q2 on b
q0 -> q2 on c
q1 -> q1 on a
q0 -> q3 on b
end_rules
start: q0
final: q2 q3
ab    # prints "accepted"
cba   # prints "rejected"
aaa   # prints "rejected"
aaab  # prints "accepted"
```