1.17. Programming Exercises

1. Implement the simple methods `getNum` and `getDen` that will return the numerator and denominator of a fraction.

2. In many ways it would be better if all fractions were maintained in lowest terms right from the start. Modify the constructor for the `Fraction` class so that `GCD` is used to reduce fractions immediately. Notice that this means the `__add__` function no longer needs to reduce. Make the necessary modifications.

3. Implement the remaining simple arithmetic operators (`__sub__` , `__mul__` , and `__truediv__`).

4. Implement the remaining relational operators (`__gt__` , `__ge__` , `__lt__` , `__le__` , and `__ne__`)

5. Modify the constructor for the fraction class so that it checks to make sure that the numerator and denominator are both integers. If either is not an integer the constructor should raise an exception.

6. In the definition of fractions we assumed that negative fractions have a negative numerator and a positive denominator. Using a negative denominator would cause some of the relational operators to give incorrect results. In general, this is an unnecessary constraint. Modify the constructor to allow the user to pass a negative denominator so that all of the operators continue to work properly.

7. Research the `__radd__` method. How does it differ from `__add__` ? When is it used? Implement `__radd__` .

8. Repeat the last question but this time consider the `__iadd__` method.

9. Research the `__repr__` method. How does it differ from `__str__` ? When is it used? Implement `__repr__` .

sionQuestions.html)

10. Research other types of gates that exist (such as NAND, NOR, and XOR). Add them to the circuit hierarchy. How much additional coding did you need to do?

11. The most simple arithmetic circuit is known as the half-adder. Research the simple half-adder circuit. Implement this circuit.

12. Now extend that circuit and implement an 8 bit full-adder.

13. The circuit simulation shown in this chapter works in a backward direction. In other words, given a circuit, the output is produced by working back through the input values, which in turn cause other outputs to be queried. This continues until external input lines are found, at which point the user is asked for values. Modify the implementation so that the action is in the forward direction; upon receiving inputs the circuit produces an output.

14. Design a class to represent a playing card. Now design a class to represent a deck of cards. Using these two classes, implement a favorite card game.

15. Find a Sudoku puzzle in the local newspaper. Write a program to solve the puzzle.

(DiscussionQuestions.html)

(../AlgorithmAnalysis/toctree.html)

user not logged in

sionQuestions.html)