Alex Elko

Mr. Kim

Adv. Computer Science

November 5, 2019

Advanced PS: Fractions

PW. 1.

1. gcd ( 12 , 18 ) = 6
   1. 12, 18
   2. 12, 18-12
   3. 12, 6
   4. 6, 12
   5. 6, 12-6
   6. 6, 6
2. gcd ( 18 , 54 ) = 18
   1. 18, 54
   2. 18, 54-18
   3. 18, 36
   4. 18, 36-18
   5. 18,18
3. gcd ( 15 , 24 ) = 3
   1. 15, 24
   2. 15, 24-15
   3. 15, 9
   4. 9, 15
   5. 9, 15-9
   6. 9, 6
   7. 6, 9
   8. 6, 9-6
   9. 6, 3
   10. 3, 6
   11. 3, 6-3
   12. 3, 3
4. gcd ( 23 , 51 ) = 1
   1. 23, 51
   2. 23, 51-23
   3. 23, 28
   4. 23, 28-23
   5. 23, 5
   6. 5, 23
   7. 5, 23-5
   8. 5, 18
   9. 5, 18-5
   10. 5, 13
   11. 5, 13-5
   12. 5, 8
   13. 5, 8-5
   14. 5, 3
   15. 3, 5
   16. 3, 5-3
   17. 3, 2
   18. 2, 3
   19. 2, 3-2
   20. 2, 1
   21. 1, 2
   22. 1, 2-1
   23. 1,1

PW. 2.

* When a and b wind up being the same value after the reduction process.

PW. 3.

* If b is greater than a, subtract a from b.
* Continue until either a == b or a is greater than b.
* If a is greater than b, then switch a and b and then subtract a from b.
* Repeat process until a == b.
* Both a and b are gcd.

PW. 4. Iterative gcd

def find\_gcd(a,b):

a,b = abs(a),abs(b)

if a == 0 and b == 0:

return a

if a == 0:

return b

if b == 0:

return a

while a != b:

if b > a:

b -= a

else:

a,b = b,a

b -=a

return a

PW. 5. Recursive gcd

def recur\_gcd(a,b):

a,b = abs(a),abs(b)

if a == 0 and b == 0:

return a

if a == 0:

return b

if b == 0:

return a

if b > a:

return recur\_gcd(a, b-a)

if a > b:

return recur\_gcd(b, a-b)

return a

TEST CASES:

* 1, 0 (0, 1)
* -1, 0 (0, -1)
* MAX\_INT, MAX\_INT-1
* 1 through 997 (1, 998 causes a recursion depth error)

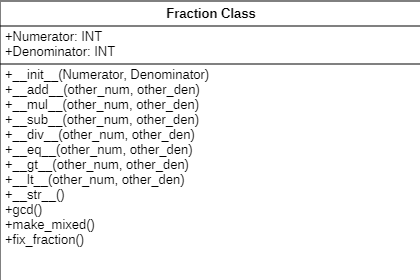
PW. B. 1:

1. 1/2 + 1/3
   1. 1/2 + 1/3
   2. 1/2×3/3 + 1/3×2/2
   3. 3/6 + 2/6
   4. 5/6
2. 1/4 + 3/5
   1. 1/4 + 3/5
   2. 1/4×5/5 + 3/5×4/4
   3. 5/20 + 12/20
   4. 17/20
3. 2/3 + 2/5
   1. 2/3 + 2/5
   2. 2/3×5/5 + 2/5×3/3
   3. 10/15 + 6/15
   4. 16/15
4. 3/4 + 5/8
   1. 3/4 + 5/8
   2. 3/4×2/2 + 5/8
   3. 6/8 + 5/8
   4. 11/8
5. 5/6 + 2/9
   1. 5/6 + 2/9
   2. 5/6×3/3 + 2/9×2/2
   3. 15/18 + 4/18
   4. 19/18
6. a/b + c/d
   1. a/b + c/d
   2. a/b×d/d + c/d×b/b
   3. (a d + b c)/(b d)

PW. B. 2.

* Multiply first fraction by base/base of second fraction
* Multiply second fraction by base/base of first fraction
* Add numerator because denominators are now the same
* Simplify using gcd

A. 1. Draw UML

* 

A. 2. Use/test case

* 1/1 + 2/2, 1/1 - 2/2
* 1/0 + 1/1
* MAXINT/MAXINT + 1/MAXINT
* 1/0 \* 1/1, 1/1 / 0/1
* 2/3 + 1/2, 2/3 – 1/2, 2/3 \* 1/2, 2/3 / 1/2
* 1/0 = 1/0
* 4/7 + 3/5
* 16/9 \* 3/2
* 191/36 \* 100/66

A.3.

* Subtraction is just calling addition with a negative numerator.
* Division is just calling multiplication with the numerator and denominator flipped.

A.4

class Fraction():

num = None

den = None

def \_\_init\_\_(self, num, den):

#makes num and den 'nice' and adds them as members

num, den = self.fix\_fraction(num,den)

self.num = num

self.den = den

def \_\_str\_\_(self):

#prints stored fraction with proper formatting

return f'fraction is {self.make\_mixed(self.num, self.den)}'

def make\_mixed(self, a, b):

#makes mixed by taking den out of num until num is less than den

whole = 0

while a >= b:

a -= b

whole += 1

if whole > 0:

if a == 0:

#returns whole number if completly factors out

return f'{whole}'

return f'{whole} {a}/{b}'

return f'{a}/{b}'

def gcd(self,a, b):

#uses gcd from earlier in problem set

#finds gcd by taking a out of b

a,b = abs(a),abs(b)

if a == 0 and b == 0:

return a

if a == 0:

return b

if b == 0:

return a

while a != b:

if b < a:

a,b = b,a

b -= a

return a

def fix\_fraction(self,a,b):

#makes a double negative num and den both pos

#moves negative from den to num

if b != abs(b) and a != abs(a):

b = abs(b)

a = abs(a)

if b != abs(b) and a == abs(a):

b = abs(b)

a = -a

return a,b

def \_\_add\_\_(self,a,b):

#makes dens equal and adds nums

a,b = self.fix\_fraction(a,b)

num = self.num\*b + a\*self.den

den = self.num\*b

div = self.gcd(num,den)

num //= div

den //= div

return self.make\_mixed(num,den)

def \_\_sub\_\_(self,a,b):

#calls addition with negative num

return self.\_\_add\_\_(-a,b)

def \_\_mul\_\_(self,a,b):

#numtiplies nums and dens and reduces

a,b= self.fix\_fraction(a,b)

num = self.num \* a

den = self.den \* b

div = self.gcd(num, den)

num //= div

den //= div

return self.make\_mixed(num,den)

def \_\_div\_\_(self,a,b):

#calls mul with flipped fraction

return self.\_\_mul\_\_(b,a)

def \_\_eq\_\_(self,a,b):

#checks for equality after dens are same

a,b = self.fix\_fraction(a,b)

if self.num \* b == a \*self.den:

return True

return False

def \_\_gt\_\_(self,a,b):

#checks if input is greater than stored after dens are same

a,b = fix\_fraction(a,b)

return True if a\*self.den > self.num\*b else False

def \_\_lt\_\_(self,a,b):

#checks if input is less than stored after dens are same

a,b = fix\_fraction(a,b)

return True if a\*self.den < self.num\*b else False