euclid.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# euclid.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

def gcd(i, j):

    if (i == 0) and (j == 0):
        raise ZeroDivisionError(
            'gcd(i,j) is undefined if i and j are 0')
    i = abs(i)
    j = abs(j)
    while j != 0:  # Euclid's algorithm
        i, j = j, i%j
    return i

#-----------------------------------------------------------------------

def lcm(i, j):

    if (i == 0) or (j == 0):
        raise ZeroDivisionError(
            'lcm(i,j) is undefined if i or j is 0')
    i = abs(i)
    j = abs(j)
    return (i // gcd(i, j)) * j
```


euclidclient3.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# euclidclient3.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import euclid

def main():

    try:
        line = input('Enter the first integer: ')
        i = int(line)

        line = input('Enter the second integer: ')
        j = int(line)

        my_gcd = euclid.gcd(i, j)
        print('gcd:', my_gcd)

        my_lcm = euclid.lcm(i, j)
        print('lcm:', my_lcm)

    except ValueError:
        print('Error: Not an integer', file=sys.stderr)
        sys.exit(1)
    except EOFError:
        print('Error: Missing integer', file=sys.stderr)
        sys.exit(1)
    except ZeroDivisionError as ex:
        print(str(ex), file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

intmath/__init__.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
```


intmath/euclid.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----------------------------------------------------------------------
4: # euclid.py
5: # Author: Bob Dondero
6: #-----------------------------------------------------------------------
7:
8: def gcd(i, j):
9:
10:     if (i == 0) and (j == 0):
11:         raise ZeroDivisionError(
12:             'gcd(i,j) is undefined if i and j are 0')
13:
14:     i = abs(i)
15:     j = abs(j)
16:     while j != 0:  # Euclid's algorithm
17:         i, j = j, i%j
18:     return i
19:
20: #-----------------------------------------------------------------------
21:
22: def lcm(i, j):
23:
24:     if (i == 0) or (j == 0):
25:         raise ZeroDivisionError(
26:             'lcm(i,j) is undefined if i or j is 0')
27:
28:     i = abs(i)
29:     j = abs(j)
30:     return (i // gcd(i, j)) * j
```

intmath/fibonacci.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fibonacci.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

def fib(num):

    if num < 0:
        raise ValueError()
    if num == 0:
        return 0
    if num == 1:
        return 1

    # The Wikipedia "Fibonacci Numbers" article describes some
    # approaches that are more efficient.

    second_prev = 0
    first_prev = 1
    while num > 1:
        current = second_prev + first_prev
        second_prev = first_prev
        first_prev = current
        num -= 1

    return current
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

euclidclient4.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# euclidclient4.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import intmath.euclid

def main():

    try:
        line = input('Enter the first integer: ')
        i = int(line)

        line = input('Enter the second integer: ')
        j = int(line)

        my_gcd = intmath.euclid.gcd(i, j)
        print('gcd:', my_gcd)

        my_lcm = intmath.euclid.lcm(i, j)
        print('lcm:', my_lcm)

    except ValueError:
        print('Error: Not an integer', file=sys.stderr)
        sys.exit(1)
    except EOFError:
        print('Error: Missing integer', file=sys.stderr)
        sys.exit(1)
    except ZeroDivisionError as ex:
        print(str(ex), file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```


fibclient.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fibclient.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import intmath.fibonacci

def main():

    try:
        for i in range(10):
            print('fib(' + str(i) + ') = ' +
                str(intmath.fibonacci.fib(i)))

    except ValueError:
        print('Error: Not a positive integer', file=sys.stderr)
        sys.exit(1)

if __name__ == '__main__':
    main()
```

fractionprelim.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fractionprelim.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import euclid

#-----------------------------------------------------------------------

class Fraction:

    def __init__(self, num=0, den=1):
        if den == 0:
            raise ZeroDivisionError('Denominator cannot be 0')
        self._num = num
        self._den = den
        self._normalize()

    def _normalize(self):
        if self._den < 0:
            self._num *= -1
            self._den *= -1
        if self._num == 0:
            self._den = 1
        else:
            gcden = euclid.gcd(self._num, self._den)
            self._num //= gcden
            self._den //= gcden

    def to_string(self):
        if self._den == 1:
            return str(self._num)
        return '%d/%d' % (self._num, self._den)

    def hash_code(self):
        return hash((self._num, self._den))   # Use tuple's __hash__().

    def equals(self, other):
        return (self._num == other._num) and (self._den == other._den)

    def compare_to(self, other):
        if (self._num * other._den) < (other._num * self._den):
            return -1
        if (self._num * other._den) > (other._num * self._den):
            return 1
        return 0

    def negate(self):
        return Fraction(-self._num, self._den)

    def add(self, other):
        new_num = (self._num * other._den) + (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

    def subtract(self, other):
        new_num = (self._num * other._den) - (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

    def multiply(self, other):
        new_num = self._num * other._num
        new_den = self._den * other._den
```

fractionprelim.py (Page 2 of 2)

```
        return Fraction(new_num, new_den)

    def divide(self, other):
        new_num = self._num * other._den
        new_den = self._den * other._num
        return Fraction(new_num, new_den)
```

fractionprelimclient.py (Page 1 of 2)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fraction1client.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import fractionprelim as fraction

def main():

    try:
        line = input('Numerator 1: ')
        num1 = int(line)
        line = input('Denominator 1: ')
        den1 = int(line)
        line = input('Numerator 2: ')
        num2 = int(line)
        line = input('Denominator 2: ')
        den2 = int(line)

        frac1 = fraction.Fraction(num1, den1)
        print('frac1:', frac1.to_string())

        frac2 = fraction.Fraction(num2, den2)
        print('frac2:', frac2.to_string())

        print('frac1 hashcode:', frac1.hash_code())

        if frac1.equals(frac2):
            print('frac1 equals frac2')
        if not frac1.equals(frac2):
            print('frac1 does not equal frac2')

        comparison = frac1.compare_to(frac2)
        if comparison < 0:
            print('frac1 is less than frac2')
        if comparison > 0:
            print('frac1 is greater than frac2')
        if comparison <= 0:
            print('frac1 is less than or equal to frac2')
        if comparison >= 0:
            print('frac1 is greater than or equal to frac2')

        frac3 = frac1.negate()
        print('-frac1:', frac3.to_string())

        frac3 = frac1.add(frac2)
        print('frac1 + frac2:', frac3.to_string())

        frac3 = frac1.subtract(frac2)
        print('frac1 - frac2:', frac3.to_string())

        frac3 = frac1.multiply(frac2)
        print('frac1 * frac2:', frac3.to_string())

        frac3 = frac1.divide(frac2)
        print('frac1 / frac2:', frac3.to_string())

    except Exception as ex:
        print(str(ex), file=sys.stderr)

#-----------------------------------------------------------------------

```

fractionprelimclient.py (Page 2 of 2)

```
if __name__ == '__main__':
    main()
```

euclidstrong.py (Page 1 of 1)

```
#!/usr/bin/env python

#-----------------------------------------------------------------------
# euclidstrong.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

def gcd(i, j):

    if not isinstance(i, int) or not isinstance(j, int):
        raise TypeError('gcd() arguments must be integers')

    if (i == 0) and (j == 0):
        raise ZeroDivisionError(
            'gcd(i,j) is undefined if i and j are 0')

    i = abs(i)
    j = abs(j)
    while j != 0:  # Euclid's algorithm
        i, j = j, i%j
    return i

#-----------------------------------------------------------------------

def lcm(i, j):

    if not isinstance(i, int) or not isinstance(j, int):
        raise TypeError('lcm() arguments must be integers')

    if (i == 0) or (j == 0):
        raise ZeroDivisionError(
            'lcm(i,j) is undefined if i or j is 0')

    i = abs(i)
    j = abs(j)
    return (i // gcd(i, j)) * j
```