```
 1  import check
 2
 3  JANUARY = 1
 4  FEBRUARY = 2
 5  MARCH = 3
 6  APRIL = 4
 7  MAY = 5
 8  JUNE = 6
 9  JULY = 7
10  AUGUST = 8
11  SEPTEMBER = 9
12  OCTOBER = 10
13  NOVEMBER = 11
14
15  SAJN = 30 #September, April, June and November
16  REST = 31
17  FEB = 28
18
19  XMASDAY = 25
20  XMASMON = 12
21
22  def is_leap_year(n):
23    '''
24    Returns true if and only if year n is a leap year
25    We ignore the British Act of 1751.
26
27    is_leap_year: Nat -> Bool
28    '''
29    if n % 400 == 0:
30      return True
31    elif n % 100 == 0:
32      return False
33    elif n % 4 == 0:
34      return True
35    return False
36
37
38  def day_in_year(d, m, y):
39    '''
40    Returns what day of the year it is given the current day (d),
```

```
41      month (m) and year (y)
42
43      day_in_year: Nat Nat Nat -> Nat
44      Requires:
45         The day given by d, m, y is valid.
46
47      Examples:
48         day_in_year(1, 1, 2000) => 1
49         day_in_year(31, 12, 2000) => 366
50         day_in_year(31, 12, 2001) => 365
51      '''
52      cur_day = 0
53      if m > JANUARY:
54        cur_day += REST
55      if m > FEBRUARY:
56        cur_day += FEB
57        if is_leap_year(y):
58          cur_day += 1
59      if m > MARCH:
60        cur_day += REST
61      if m > APRIL:
62        cur_day += SAJN
63      if m > MAY:
64        cur_day += REST
65      if m > JUNE:
66        cur_day += SAJN
67      if m > JULY:
68        cur_day += REST
69      if m > AUGUST:
70        cur_day += REST
71      if m > SEPTEMBER:
72        cur_day += SAJN
73      if m > OCTOBER:
74        cur_day += REST
75      if m > NOVEMBER:
76        cur_day += SAJN
77      cur_day += d
78      return cur_day
79
80    def days_until_christmas(day, month, year):
```

```
81      '''
82      Returns the number of days until Christmas given the day month and ↵
        ↳ year
83      in the Gregorian calendar
84
85      days_until_christmas: Nat Nat Nat -> Nat
86      Requries: (day, month, year) is a valid Gregorian calendar date
87
88      Examples:
89         days_until_christmas(25,12,2020) => 0
90         days_until_christmas(1,1,2020)   => 359
91      '''
92      if month == XMASMON and day > XMASDAY:
93        return REST - day + day_in_year(XMASDAY, XMASMON, year+1)
94      return day_in_year(XMASDAY, XMASMON, year) - \
95             day_in_year(day, month, year)
96
97    ##Examples
98    check.expect("Christmas Day", days_until_christmas(25,12,2020), 0)
99    check.expect("With Leap", days_until_christmas(1,1,2020), 359)
100
101   ##Tests
102   check.expect("Boxing Day with Leap",
103                days_until_christmas(26,12,2019), 365)
104   check.expect("Boxing Day +1",
105                days_until_christmas(27,12,2019), 364)
106   check.expect("Boxing Day +2",
107                days_until_christmas(28,12,2019), 363)
108   check.expect("Boxing Day +3",
109                days_until_christmas(29,12,2019), 362)
110   check.expect("Boxing Day +4",
111                days_until_christmas(30,12,2019), 361)
112   check.expect("Boxing Day +5",
113                days_until_christmas(31,12,2019), 360)
114   check.expect("Boxing Day +6",
115                days_until_christmas(1,1,2020), 359)
116   check.expect("Boxing Day with leap",
117                days_until_christmas(26,12,2019), 365)
118   check.expect("Boxing Day",
119                days_until_christmas(26,12,2020), 364)
```

```
120   check.expect("Christmas Eve",
121                days_until_christmas(24,12,2020), 1)
122   check.expect("Dec 1",
123                days_until_christmas(1,12,2020), 24)
124   check.expect("Nov 30",
125                days_until_christmas(30,11,2020), 25)
126   check.expect("Dec 31 With Leap",
127                days_until_christmas(31,12,2019), 360)
128   check.expect("Dec 30 With Leap",
129                days_until_christmas(30,12,2019), 361)
130   check.expect("Jan 1 Without Leap",
131                days_until_christmas(1,1,2021), 358)
132   check.expect("Dec 31 Without Leap",
133                days_until_christmas(31,12,2068), 359)
134   check.expect("Dec 30 Without Leap",
135                days_until_christmas(30,12,2068), 360)
136   check.expect("Random",
137                days_until_christmas(3,9,2020), 113)
138
```

```
1   import check
2
3
4   def is_prime(p, d):
5     '''
6     Returns True if and only if p has no divisor from 2 to d
7     and False otherwise.
8
9     is_prime: Nat Nat -> Bool
10    Requires:
11       p > 1
12       0 < d <= p
13    '''
14    if d == 1:
15      return True
16    if p % d == 0:
17      return False
18    return is_prime(p, d-1)
19
20  def squarefree(n, d):
21    '''
22    Returns True if n is square free (testing from divisor d down)
23    and False otherwise
24
25    squarefree: Nat Nat -> Bool
26    Requires: 0 < d < n
27    '''
28    if d <= 1:
29      return True
30    if n % d**2 == 0 and is_prime(d, d-1):
31      return False
32    return squarefree(n, d-1)
33
34  def num_prime_divisors(n, d):
35    '''
36    Returns a count of all prime divisors of n between 1 and d.
37
38    num_prime_divisors: Nat Nat -> Nat
39    Requires: 0 < d < n
40    '''
```

```
41      if d <= 1:
42        return 0
43      return int(n % d == 0 and is_prime(d, d-1)) + num_prime_divisors(n, ↵
      ↳ d-1)
44
45    def mobius(n):
46      '''
47      Returns the value of the mobius function of n. This is:
48        *1 if the number has an even number of prime divisors and is ↵
      ↳   squarefree
49        *-1 if the number has an odd number of prime divisors and is ↵
      ↳   squarefree
50        *0 if the number is not squarefree
51
52      mobius: Nat -> Int
53      Requires: 0 < n
54
55      Examples:
56        mobius(1) => 1
57        mobius(2) => -1
58        mobius(4) => 0
59      '''
60      if not squarefree(n, n//2):
61        return 0
62      if num_prime_divisors(n, n) % 2 == 0:
63        return 1
64      return -1
65
66    ##Examples:
67    check.expect("Test smallest", mobius(1), 1)
68    check.expect("Test prime", mobius(2), -1)
69    check.expect("Test prime", mobius(3), -1)
70    check.expect("Test square", mobius(4), 0)
71
72    ##Tests:
73    check.expect("Test even divisors", mobius(6), 1)
74    check.expect("Test not square free", mobius(12), 0)
75    check.expect("Test prime", mobius(5), -1)
76    check.expect("Test prime", mobius(101), -1)
77    check.expect("Test composite", mobius(91), 1)
```

```
78
79  ##Personal Large Tests
80  check.expect("Test composite", mobius(13*17), 1)
81  check.expect("Test three primes", mobius(2*13*17), -1)
82  check.expect("Test four primes", mobius(2*3*5*7), 1)
83  check.expect("Test not square free", mobius(2*3*5*7*5), 0)
84
```

```
1   import check
2   import math
3
4
5   def iterated_continued_fraction(n, iters):
6     '''
7     Returns the continued fraction expansion of n up to iters iterations
8     after the first value
9
10    iterated continued_fraction: Float Nat -> Str
11    Requires: n > 0.0
12
13    Examples:
14       iterated_continued_fraction(math.e, 11)
15          => "1,2,1,1,4,1,1,6,1,1,8,...]"
16       iterated_continued_fraction(math.pi, 0) => "...]"
17    '''
18    if iters == 0:
19      return "...]"
20    integer_part = math.floor(n)
21    mantissa = n - integer_part
22    return str(integer_part) + "," + \
23           iterated_continued_fraction(1/mantissa, iters-1)
24
25  def continued_fraction(n, iters):
26    '''
27    Returns the continued fraction expansion of n up to iters iterations
28
29    continued_fraction: Float Nat -> Str
30    Requires:
31       n > 0.0 and is irrational
32       iters > 0
33
34    Examples:
35       continued_fraction(math.e, 12)
36          => "[2;1,2,1,1,4,1,1,6,1,1,8,...]"
37       continued_fraction(math.pi, 1) => "[3;...]"
38    '''
39    integer_part = math.floor(n)
40    mantissa = n - integer_part
```

```
41      return "[" + str(integer_part) + ";" + \
42             iterated_continued_fraction(1/mantissa, iters-1)
43
44  ##Examples
45  check.expect("Test", continued_fraction(math.e, 12),
46               "[2;1,2,1,1,4,1,1,6,1,1,8,...]")
47  check.expect("Test", continued_fraction(math.pi, 1), "[3;...]")
48
49  ##Tests:
50  check.expect("Test", continued_fraction(3*math.pi/4, 10),
51               "[2;2,1,4,5,6,24,3,3,3,...]")
52  check.expect("Test sqrt 2", continued_fraction(math.sqrt(2), 7),
53               "[1;2,2,2,2,2,2,...]")
54  check.expect("Test sqrt 7", continued_fraction(math.sqrt(7), 15),
55               "[2;1,1,1,4,1,1,1,4,1,1,1,4,1,1,...]")
56  check.expect("Test less than 0", continued_fraction(math.log(2), 15),
57               "[0;1,2,3,1,6,3,1,1,2,1,1,1,1,3,...]")
```