Extended Euclidean Algorithm

The Euclidean algorithm works by successively dividing one number (we assume for convenience they are both positive) into another and computing the integer quotient and remainder at each stage. This produces a strictly decreasing sequence of remainders, which terminates at zero, and the last nonzero remainder in the sequence is the gcd. This works because if $a = bq+r$ then $\gcd(a, b) = \gcd(b, r)$ so we are continually trading in the gcd of a pair for the gcd of a smaller pair. At the last step, we have $\gcd(\overline{r}, r) = \gcd(r, 0) = r$ where r is the final nonzero remainder and $\overline{r}$ is the remainder preceding it in the sequence.

Here is a simple iterative implementation of the algorithm in Python:

```
def gcd(a,b):
  while b:
    a,b = b, a % b
  return a
```

Note that this works even if $a < b$, since then its first step will be to interchange a and b, after which the reductions will take place as usual. Here is a recursive implementation of the same algorithm, also in Python:

```
def gcd(a,b):
  if b == 0:
    return a
  else:
    return gcd(b, a % b)
```

and again if $a < b$ this first interchanges a and b and then proceeds as usual, so as in the preceding example the order of the arguments does not matter.

Now by working backwards with all the quotients and remainders, one can easily find integers x, y such that $g = ax + by$ where $g = \gcd(a, b)$. We'd like to modify the first program above to return x and y as well as g. In fact, not only does the algorithm produce a sequence of remainders, but for *each* remainder it produces integers x and y so that $r = ax + by$. Thus we obtain sequences of x's and y's. Supposing that the previous remainder $\overline{r} = a\overline{x} + b\overline{y}$, at the next stage of the algorithm we would divide r into the previous remainder $\overline{r}$ to obtain an integer quotient q and a new remainder, and we have

$$\overline{r} = qr + (\texttt{new}\ r)$$

so by solving for the new remainder and substituting we obtain

$$\texttt{new}\ r = \overline{r} - qr = (ax + by) - q(a\overline{x} + b\overline{y}) = a(x - q\overline{x}) + b(y - q\overline{y}).$$

This proves that the x and y sequences must satisfy the two-stage recursion:

$$\texttt{new}\ x = x - q\overline{x}; \qquad \texttt{new}\ y = y - q\overline{y}$$

where $\overline{x}$, $\overline{y}$ are the predeccessors of x, y in the x and y sequences.

It remains to work out the base case of the recursion. But $a = a(1) + b(0)$ and $b = a(0) + b(1)$, so we should start by initially taking $\overline{x} = 1$, $\overline{y} = 0$ and $x = 0$, $y = 1$. Note that on termination the values $\overline{x}$ and $\overline{y}$ are the ones desired, since the x and y sequences have gone one stage too far (as has the r sequence).

These facts enable us to construct the following Python code:

```
def xgcd(a,b):
  prevx, x = 1, 0;  prevy, y = 0, 1
  while b:
      q = a/b
      x, prevx = prevx - q*x, x
      y, prevy = prevy - q*y, y
      a, b = b, a % b
  return a, prevx, prevy
```

which extends the iterative code given previously for the gcd. This new program returns the triple g, x, y instead of just g.

Note the use of the semicolon which allows us to put two Python multiple assignment statements on the same line.

Exercise. Write a recursive implementation of xgcd using the above analysis.