Week 3: Homework Solutions

3.

```
a = int(input("a: "))
b = int(input("b: "))
while a != b:
    a, b = min(a, b), max(a, b) - min(a, b)
print("The greatest common divisor is", a)
```

An example of a faster algorithm is Euclid's algorithm by division:

```
a = int(input("a: "))
b = int(input("b: "))
# Swap a and b if necessary, so that a <= b
a, b = min(a, b), max(a, b)
while b % a != 0:
    a, b = b % a, a
print("The greatest common divisor is", a)
```

This implementation uses of the Python modulus operator (%). For large values of *a* and *b* Euclid's algorithm by division tends to be faster than our subtraction algorithm, since it performs fewer operations.

4. Here are two solutions. The first is a straight translation of the formulas on the wikipedia page.

```
# To calculate the Gauss-Legendre approximation to pi. (1)
a, b, t, p = 1, 2**(-0.5), 1/4, 1
count = 1
# We will just run 10 iterations to see what we get
while count <= 10:
    a_new = (a + b)/2
    b_new = (a*b)**(0.5)
    t_new = t - p*(a - a_new)**2
    p_new = 2*p
    # now update the values to the new ones
    a, b, t, p = a_new, b_new, t_new, p_new
    pi_estimate = (a + b)**2 / (4*t)
    print("Estimate", count, "is", pi_estimate)
    count = count + 1
```

If you look carefully you will see that the only new variable on the right hand-side of the definitions is `a_new` in the definition of `t_new`, so substituting the definition of `a_new` in to `t_new` gives us a set of definitions that we can do with a single swoop in a parallel assignment.

```
# To calculate the Gauss-Legendre approximation to pi. (2)
a, b, t, p = 1, 2**(-0.5), 1/4, 1
count = 1
# We will just run 10 iterations to see what we get
while count <= 10:
    a, b, t, p = (a + b)/2, (a*b)**(0.5), t - p*((a - b)/2)**2, 2*p
    pi_estimate = (a + b)**2 / (4*t)
    print("Estimate", count, "is", pi_estimate)
    count = count + 1
```

When you run these programs you see that the accuracy of the approximation quickly exceeds the accuracy to which Python floats are presented. There are variable types in Python which are better suited than floats for higher precision work.