# Loop Invariant Examples



**Problems:**

Exercise 0

@param a list of n doubles
@return the average of the list
average(double[ ] a)
  // Yes, this should be as easy as it sounds

Exercise 1

@param a pre-computed average
@param the number of values used in the average
@param a list of n doubles
@return a new weighted average of the old average combined with the values of array a
runningAverage(double avg, int count, double[ ] a)
  // There are 3 components to an average:  valuesum, valuecount, average.
  // The formula you are used to is average = valuesum / valuecount
  // When you are provided with only 2 components, it is easy to compute the third.

Exercise 2

```
@requires n >= 2
@return a list of all primes from 2 to n inclusive.
allPrimes(int n)
  // You can create an auxillary list/queue/stack if it helps
  // Remember that a prime number cannot be divided by any other number excepting 1 and itself
```

Exercise 3

Euclidian Greatest Common Divisor algorithm
(Careful, this one is a doosey)

```
@returns the greatest common divisor of a and b
int gcd(int a, int b) {
  // remember, the GCD of two numbers is the greatest number that can divide both a & b
  // without a remainder.  e.x. gcd(6, 9) = 3  {  6 divisible by [1,2,3,6] & 9 divisible by [1,3,9]  }
  // Also, don't be afraid to set conditions in your invariant.
  // e.x. { x/y = a || y==0 } is equivalent to saying { if y!=0 then x/y = a }
  // These can be useful for getting past the set-up stage of the loop with your invariant... provided
  // your conditions are sure to fall away by the end of the loop.
```

**Help:**

Exercise 0

```
@param a list of n doubles
@return the average of the list
average(double[ ] a) {
  sum = 0
  int i = 0
  while (i < a.length) {
    sum += a[ i ]
    i++
  return sum / a.length
```

Exercise 1

```
@param a pre-computed average
@param the number of value used in the average
@param a list of n doubles
@return the a new weighted average of the old average combined with the values of array a
runningAverage(double avg, int count, double[ ] a) {
  newavg = avg
  int i = 0
  while (i < a.length) {
    newcount = (count + i)
    newtotal = (newavg * newcount) + a[i]
    newavg = newtotal / (newcount + 1)
    i++
  }
  return newavg
}
```

Exercise 2

Here's some pseudo code you can follow

```
@requires n >= 2
@return a list of all primes from 2 to n inclusive.
allPrimes(int n) {
  create a list Q and fill it with the consecutive integers 2 through n inclusive.
  create an empty list P to store primes.
  p = 0
  while (p <= n) {
    obtain the next prime p by removing the first value in Q.
    put p at the end of P.
    loop through Q, eliminating numbers divisible by p.
  }
  return P
}
```

Exercise 3

```
int gcd(int a, int b) {
  int x = a;
  int y = b;
  while( y != 0 ) {
    int r = x % y;
    x = y;
    y = r;
  }
  return x;
}
```

**Solutions:**

// Disclaimer, if you think you've found an issue with a solution, please let us know!
// Us TAs are in fact capable of making mistakes on some of these complicated proofs.

Exercise 0

```
@param a list of n doubles
@return the average of the list
average(double[ ] a) {
  sum = 0
  int i = 0
  { average a[0...i-1] = sum / i }
  while (i < a.length) {
    { average a[0...i-1] = sum / i }
    sum += a[ i ]
    i++
    { average a[0...i-1] = sum / i }
  }
  { average a[0...i-1] = sum / i }
  return sum / a.length
}
```

Exercise 1

```
@param a pre-computed average
@param the number of value used in the average
@param a list of n doubles
@return the a new weighted average of the old average combined with the values of array a
runningAverage(double avg, int count, double[ ] a) {
  newavg = avg
  int i = 0
  { newavg = running average including elements a[0...i-1] }
  while (i < a.length) {
    { newavg = running average including elements a[0...i-1] }
    newcount = (count + i)
    newtotal = (newavg * newcount) + a[i]
    newavg = newtotal / (newcount + 1)
    i++
    { newavg = running average including elements a[0...i-1] }
  }
  { newavg = running average including elements a[0...i-1] }
  return newavg
}
```

Exercise 2

```
@requires n >= 2
@return a list of all primes from 2 to n inclusive.
allPrimes(int n) {
  create a list Q and fill it with the consecutive integers 2 through n inclusive.
  create an empty list P to store primes.
  p = 0
  { array P contains all primes from [2...p] }
  while (p <= n) {
    { array P contains all primes from [2...p] }
    obtain the next prime p by removing the first value in Q.
    put p at the end of P.
    loop through Q, eliminating numbers divisible by p.
    { array P contains all primes from [2...p] }
  }
  { array P contains all primes from [2...p] }
  return P
}
```

Exercise 3

```
// Notice how the OR conditions { x>Y,  x<y } carefully allow us to setup the loop without
// invalidating the main part of the invariant { X%x = y%Y && Y%x = 0 }

int gcd(int a, int b) {
  int x = a; 14
  int y = b; 6
  { X=x, Y=y }
  { inv: (X%x = y%Y && (Y%x = 0 || x>Y) ) || x < y }
  while( y != 0 ) {
    { inv: (X%x = y%Y && (Y%x = 0 || x>Y) ) || x < y }
    int r = x % y;
    x = y;
    y = r;
    { inv: (X%x = y%Y && (Y%x = 0 || x>Y) ) || x < y }
  }
  { inv: (X%x = y%Y && (Y%x = 0 || x>Y) ) || x < y }
  return x;
}
```