Gcd with the classic euclid using subtraction

```
#16 seconds for 2 numbers for 20000000 iterations
```

This algorithm works by substracting the smaller number from the bigger number until they are equal. In reality we stop right before that happens because the gcd of this pair will be kept in the last bigger number, our "a" variable.

```
<<Subtraction Algorithm>>=
def gcdSubstraction(a, b):
    while a != b:
        if (a > b):
            a = a - b
            print(a)
        else:
            b = b - a
            print(b)

    return a
@
```

Example:

gcdSubtraction(12,20)

```
12 < 20 => b = 8;
12 > 8 => a = 4;
4 < 8 => b = 4;
```

yields 4

Gcd with the division euclid

```
#19 seconds for 2 numbers for 20000000 iterations
```

This algorithm works by dividing the greater number by the smaller* and taking the remainder, we do this by using %. We do this until the b becomes 0. Our gcd value will be found in variable a. {* we know this because if we use Smaller%Greater the modulo is the smaller one and they are simply switched}

```
<<Division Algorithm>>=
def gcdDivision(a, b):
    while b != 0:
        a, b = b, a % b
        print("a = %s;"%a,"b = %s;"%b)

    return a
@
```

Example:

gcdDivision(12, 20)

```
a = 20; b = 12 % 20 = 12;
a = 12; b = 20 % 12 = 8;
a = 8; b = 12 % 8 = 4;
a = 4; b = 8 % 4 = 0;
```

yields 4

Proof(?): Let a, b be the two numbers if a % b is 0 (ex: 12, 4) the gcd will be the smaller one. so we search for pairs of this form. in addition we need to prove that if a number is greater than the other, the algorithm works for the pair (a,b-a) if a < b if a number divides completly both a and b it will also divide their difference so the divisior will fit an integer number of times in both a and b if we divide multiple times we will get to a (a,b) pair s.t a%b = 0;

Gcd for arbitrary size of numbers

```
24 seconds for a list of  2 numbers for 10000000 iterations
```

Here we do the `naive` implementation for a gcd algorithm of a list of numbers. We compute the gcd of the first elements of the list. Then the result with the next element, until we did it with all the set. As how we do the gcd, we just start from 1 -> smaller one of the pair and checks if it divides both of the numbers. The last number that divides both of them is the gcd.

```
<<Loop Algorithm>>=
def gcdLoop(inputList):
    if (len(inputList) != 0):
        gcd = inputList[0]
        for i in range(1, len(inputList)):
            if gcd > inputList[i]:
                lowBound = inputList[i]
            else:
                lowBound = gcd
            for j in range(1, lowBound + 1):
                if ((gcd % j == 0) and (inputList[i] % j == 0)):
                    print("dasd")
                    currentGcd = j
                    print("current gcd =%s;" %x`currentGcd)
            gcd = currentGcd
    return gcd
@
```

Example:

gcdLoop([60,48]):

```
current gcd = 2;
current gcd = 3;
current gcd = 4;
current gcd = 6;
```

```
current gcd = 12;
```

yields 12

Proof(?):

```
Let j = [1,min(gcd,currentListIem)]
For any two random numbers there exists at least one j that divides them
    - 1 if they are prime
    - 2 if they are even

So it should work for any valid two positive integers.
```

```
<<*>>=
<<Subtraction Algorithm>>

<<Division Algorithm>>

<<Loop Algorithm>>


print("subtraction")
print(gcdSubtraction(12,20))
print("division")
print(gcdDivision(12,20))
print("loop")
print(gcdLoop([12,20]))
@
```