WHAT IS AN ALGORITHM

- An algorithm is a description of how to systematically perform a task.
- Algorithm is a sequence of steps which are followed to achieve some objection. These steps can be considered as a recipe or a program.
- And we write a program using programming language. So goal of programming language is to describe the sequence of steps that are required and to also describe how we can pursue different sequences of steps if different things happen in between.
- Arrange 80 chairs in a Hall in 8 Rows and each row should have 10 chairs.

OUR AGENDA

- Algorithms that manipulate information

  eg: Compute numerical functions **$f(x,y) = x^y$**

- Reorganize data --- Arrange in ascending order
- Optimization --- To find shortest route

Greatest Common Divisor

Greatest Common Divisor (GCD) or Highest Common Factor (HCF) of two positive integers is the largest positive integer that divides both numbers without remainder. It is useful for reducing fractions to be in its lowest terms.

If we want to find the **gcd(m,n)**

Let ***k*** be the largest factor which divides both ***m*** and ***n***

Example-1

gcd(8,12) = 4

gcd(7,25) = 1

1 divides every number so there will be at least one common divisor for ***m*** and ***n***

How can we compute the gcd(m,n)

We need to define the systematic way to compute gcd

- Test each number in the range 1 to ***m*** to list out factors of ***m,*** must be between 1 and ***m***
- If number divides ***m*** without leaving a remainder add that number to the list of factors
- In Similar way, test each number in the range 1 to ***n*** to list out factors of ***n,*** must be between 1 and ***n***
- If number divides ***n*** without leaving a remainder add that number to the list of factors
- Find out largest factor which is present in both lists

Example-2

gcd(14,63)

Factors of 14	1, 2, 7, 14
Factors of 63	1, 3, 7, 9, 21, 63
Common Factors	1, 7
Highest Common Factor	7

Writing an algorithm for gcd(m,n)

- Use ***fm*** (factors of m) & ***fn*** (factors of n) for each list.
- For each ***i*** from 1 to ***m***, add ***i*** to ***fm*** if ***i*** divides ***m*** without leaving any remainder.
- For each ***j*** from 1 to ***n***, add ***j*** to ***fn*** if ***j*** divides ***n*** without leaving any remainder.
- Use ***cf*** (common factors) for listing common factors
- For each ***f*** in ***fm***, add ***f*** to ***cf*** if ***f*** also appears in ***fn***
- Return largest value in ***cf***

OUR FIRST PYTHON PROGRAM TO COMPUTE GCD

```
def gcd (m,n):
    fm = []
    for i in range (1,m+1):
        if (m%i) == 0:
            fm.append(i)
    fn = []
    for j in range (1, n+1):
        if (n%j) == 0:
            fn.append(j)
    cf = []
    for f in fm:
        if f in fn:
            cf.append(f)
    return(cf[-1])
```

FEW HIGHLIGHTS

- We can define a function using ***def***
- Where ***m*** & ***n*** are two arguments
- ***List*** is defined with square brackets i.e. **[ ]**
- "="is an assignment operator
- ***Range*** is a built in function, m+1 is used because if we use range (1,5) that means 1,2,3,4.
- 5 is not included in the range but to include m in the range we can give the range as ***m+1***
- % is Modulus operator.
- If remainder of ***m%i equals to zero append i to list fm*** meaning add ***i*** to list ***fm***
- We need right most element from common factors ***cf*** so we are using **return**(cf[-1]) that means return -1th element in ***cf*** which will give the right most element.

Few Points to Remember

- Use names to remember intermediate values: m, n, fm, fn, i, j, f
- Values can be single item or collections
- In above example m, n, i, j, f are single numbers
- In above example fm, fn, cf are lists of numbers
- Update list using append
- Some steps are repeated, do the same thing for each item in a list.
- Some steps are executed conditionally meaning do something if a value meets a requirement.

HOW CAN WE IMPROVE OUR FIRST PYTHON PROGRAM

- We scanned numbers from 1 to ***m*** to compute ***fm*** and again scanned numbers from 1 to ***n*** to compute ***fn***
- Why not to scan one time from 1 to ***max (m,n)*** ?
- For each ***i*** in 1 to *max****(m,n)***, add ***i*** to ***fm*** if ***i*** divides ***m*** and add ***i*** to ***fn*** if ***i*** divides ***n***

FURTHER IMPROVING OUR FIRST PROGRAM

- Why do we need to compute two lists & then compare them to compute common factors cf ?
- Let's do it in one shot
- For each ***i*** in 1 to *max****(m,n)***, if ***i*** divides ***m*** and ***i*** divides ***n***, then add ***i*** to ***cf***
- Any common factor will be less than *min****(m,n)***
- For each ***i*** in *min****(m,n)*** ,if ***i*** divided m & ***i*** also divides ***n***, add ***i*** to ***cf***

```
def gcd(m,n):
    cf = []
    for i in range (1, min(m,n)+1):
        if (m%i) == 0 and (n%i) == 0:
            cf.append(i)
    return (cf[-1])
```

Note: ***add*** is a logical operator

DO WE ACTUALLY NEED LISTS

- We only need the largest common factor.
- One will always be a common factor
- Each time we find the largest common factor we can discard the previous one
- We only need to remember largest common factor seen so far & we can return it.
- Let mrcf be Most Recent Common Factor

```
def gcd(m,n):
    for i in range (1, min(m,n)+1):
        if (m%i == 0) and (n%i == 0):
            mrcf = i
    return (mrcf)
```

CAN WE FURTHER OPTIMIZE OUR PROGRAM TO COMPUTE GCD?

- Yes! We Can
- For further optimization, as we need to find largest common factor. We can start from the end and work backwards.
- Let ***i*** run from *min****(m,n)*** to 1
- First common factor which we will encounter will always be GCD

```
def gcd(m,n):
    i = min(m,n)
    while i > 0:
        if(m%i)== 0 and (n%i) == 0:
            return(i)
        else:
            i = i-1
```

Note:

FOR LOOP is used for fixed number of repetitions & ***WHILE LOOP*** is used when you don't know about number of repetitions.