An algorithm, is an ordered set of unambiguous and well-defined instructions that performs some task and halts in finite time.

- an ordered set : you can number the steps.
- unambiguous : each instruction is clear, do-able and can be done without difficulty. (does not require creative skills)
- performs some task
- halts in finite time : algorithms terminate!

A pseudocode is a notational system in which ideas can be expressed informally during the algorithm development process.

1. saving a computed value - assignment statements

```
name ← expression
          (assign name, the value of expression)
c ← a + b
```

2. conditional operations

```
if (condition) then (activity)        if (condition) then (activity)
               else (activity)

if (sales have decreased) then (lower the price by 5%)
if (a > 0) then (x ← x + 1)
           else (x ← x - 1)
```

3. Iterative operations

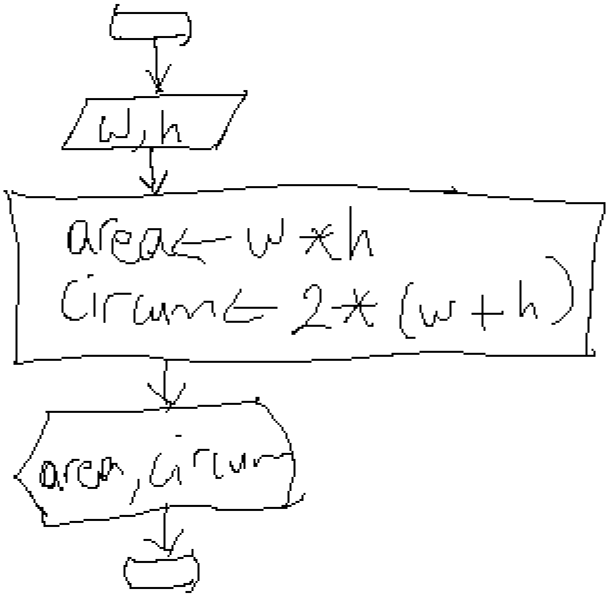
```
while (condition) do (activity)
repeat (activity) until (condition)
```

Indentation :

```
if (not raining) then ( if (temperature = hot) then (go swimming)
else (play golf)) else (watch television)

if (not raining)
   then ( if (temperature = hot)
             then (go swimming)
             else (play golf)
        )
   else (watch television)
```

Ex 1) Read the dimensions of a rectangle, print its area and circumference.

```
Read w, h
area ← w*h
circum ← 2*(w+h)
Print “The area is”, area, “and the circumference is”, circum
```

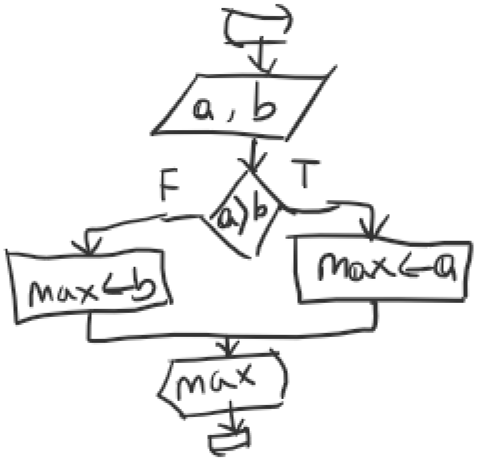


Ex 2) Read the dimensions of a cylinder, print its area and volume.

```
Read r, h
Pi ← 3.14
base ← Pi * r * r
side ← 2 * Pi * r * h
area ← 2 * base + side
volume ← base * h
Print area, volume
```

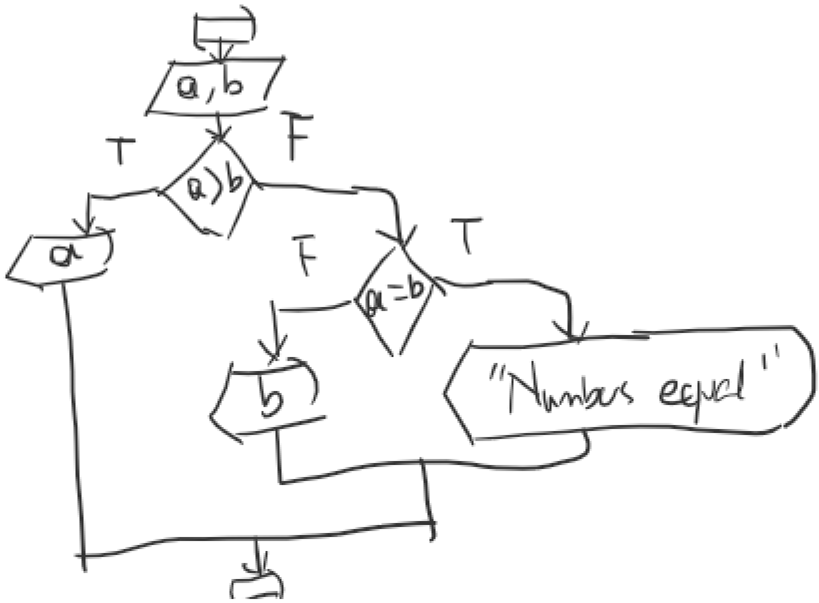
Ex 3) Read two numbers and print the larger one (assume distinct).

```
Read a, b
if( a > b )
      then( max ← a )
      else( max ← b )
Print max
```

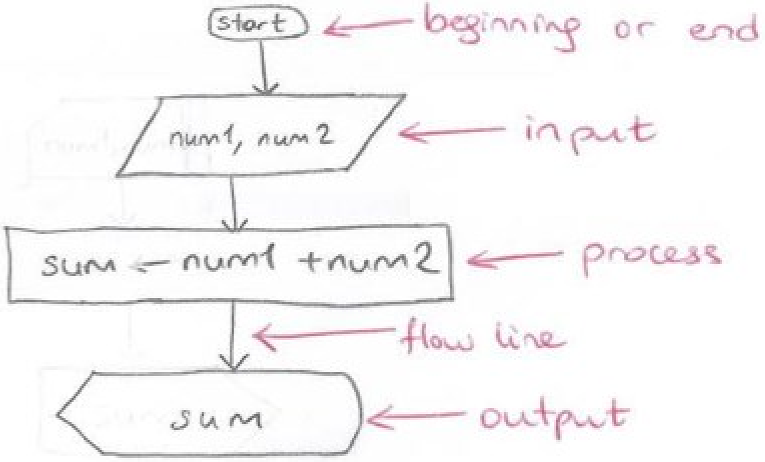


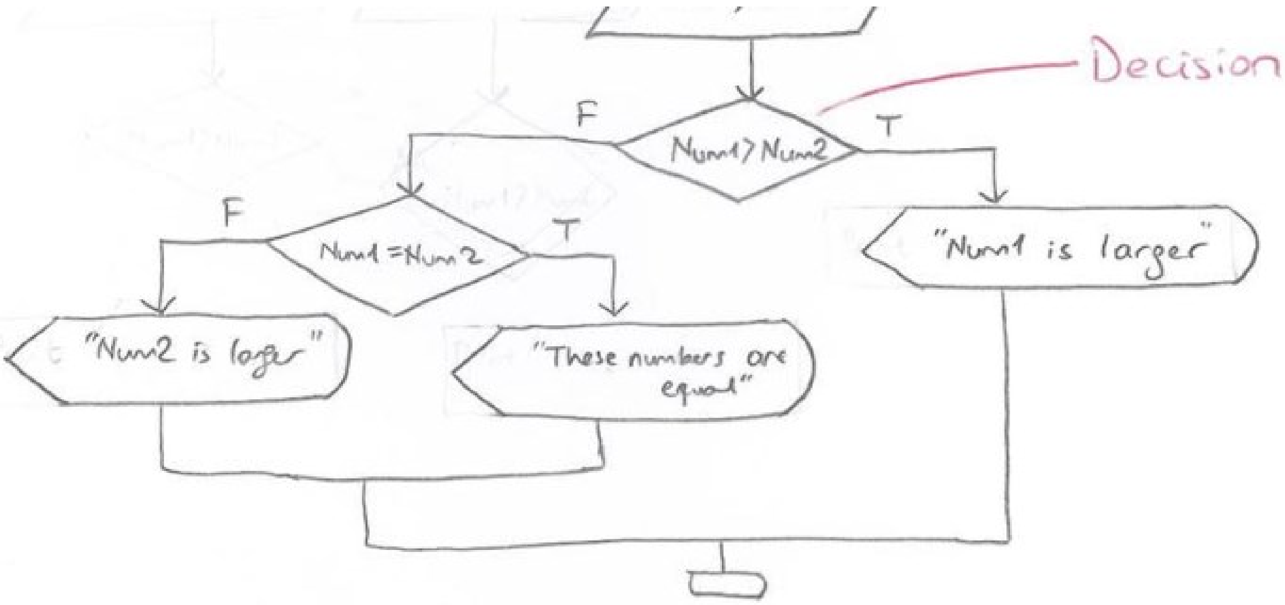
Ex 3) Read two numbers and print the larger one (no assumption). If equal, inform the user.

```
Read a, b
if( a > b )
      then( Print a )
      else( if ( a = b )
                  then( Print “Numbers are equal” )
                  else( Print b )
          )
```

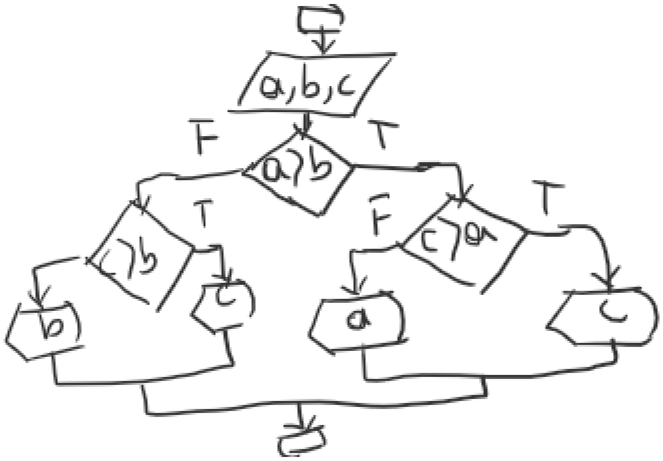


Flowchart :





Ex 4) Read three numbers and print the max. Assume distinct numbers.

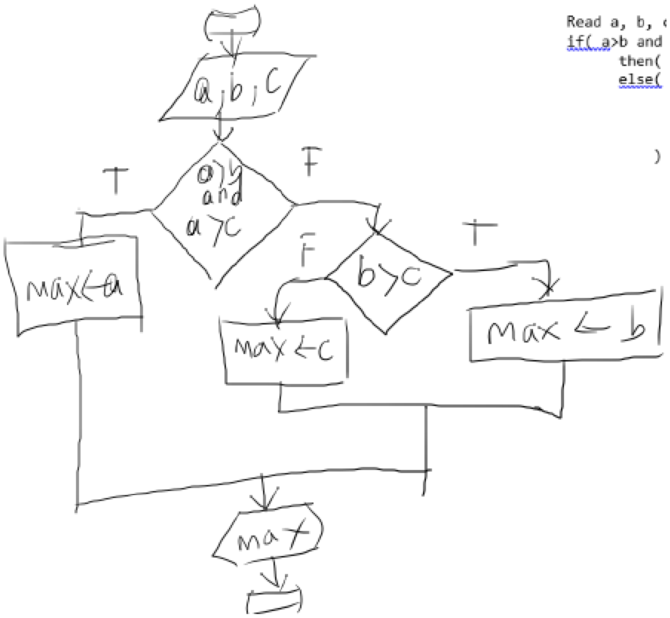


Ex 5) Read three numbers and print the max. Assume distinct numbers. Logical operators allowed.

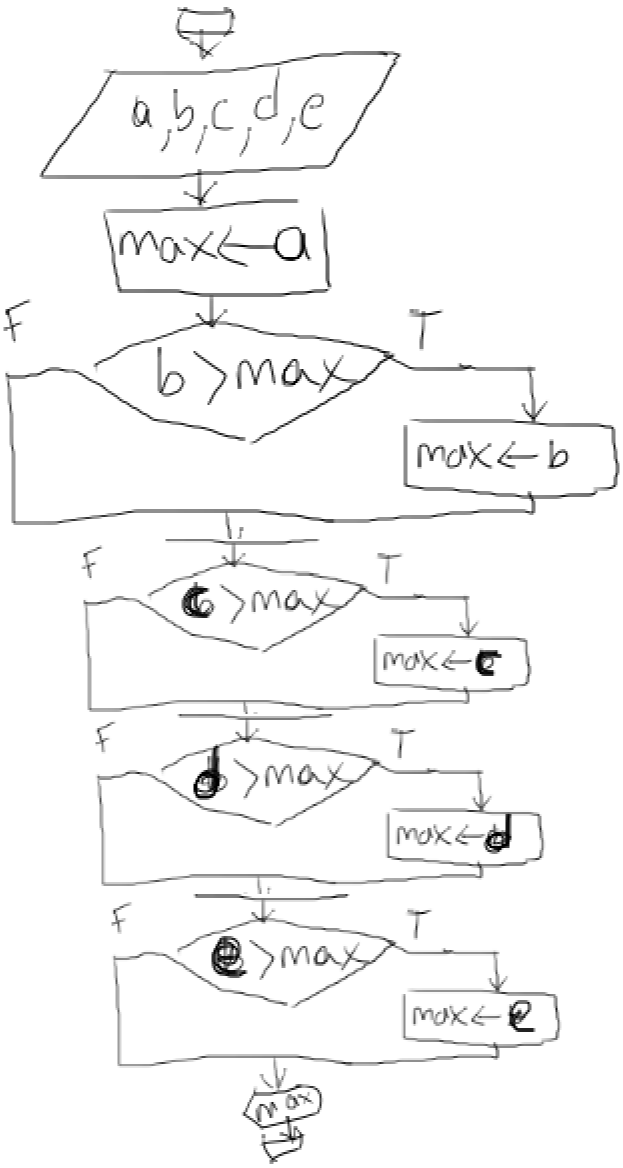
```
Read a, b, c.
if( (a>b and b>c) or (a>c and c>b) )
      then( max ← a )
if( (b>a and a>c) or (b>c and c>a) )
      then( max ← b )
if( (c>a and a>b) or (c>b and b>a) )
      then( max ← c )

simpler and more efficient:
Read a, b, c.
if( (a>b and b>c) or (a>c and c>b) )
      then( max ← a )
      else(
                  if( (b>a and a>c) or (b>c and c>a) )
                        then( max ← b )
                        else(
                              if( (c>a and a>b) or (c>b and b>a) )
                                    then( max ← c )

                            )
          )

simpler and more efficient:
Read a, b, c.
if( (a>b and b>c) or (a>c and c>b) )
      then( max ← a )
      else(
                  if( (b>a and a>c) or (b>c and c>a) )
                        then( max ← b )
                        else(
                              ~~if( (c>a and a>b) or (c>b and b>a) )~~
                                    ~~then(~~ max ← c ~~)~~

                            )
          )


simpler and more efficient:
Read a, b, c.
if( a>b and a>c )
      then( max ← a )
      else(
                  if( b>a and b>c )
                        then( max ← b )
                        else( max ← c )
          )
```

simpler and more efficient:

```
Read a, b, c.
if( a>b and a>c )
      then( max ← a )
      else(
                  if( b>c )
                        then( max ← b )
                        else( max ← c )
          )
```

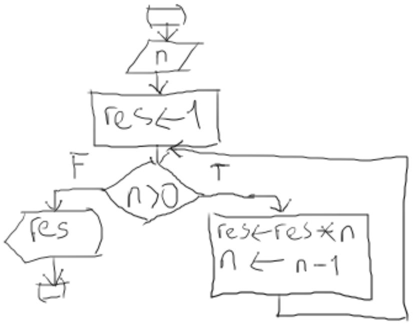


Ex 6) Read five numbers and print the max. Assume distinct numbers.



```
Read a, b, c, d e.
max ← a
if( b > max ) then ( max ← b )
if( c > max ) then ( max ← c )
if( d > max ) then ( max ← d )
if( e > max ) then ( max ← e )
Print e
```

Ex 7) Read n. Find n! [n * (n-1) * (n-2) … 2 * 1]



```
result←1
n:5
res←res*n
n←n-1
if n>0 continue
   else stop
res←res*n
n←n-1
if n>0 continue
   else stop
res←res*n
n←n-1
if n>0 continue
   else stop
res←res*n
n←n-1
if n>0 continue
   else stop
```

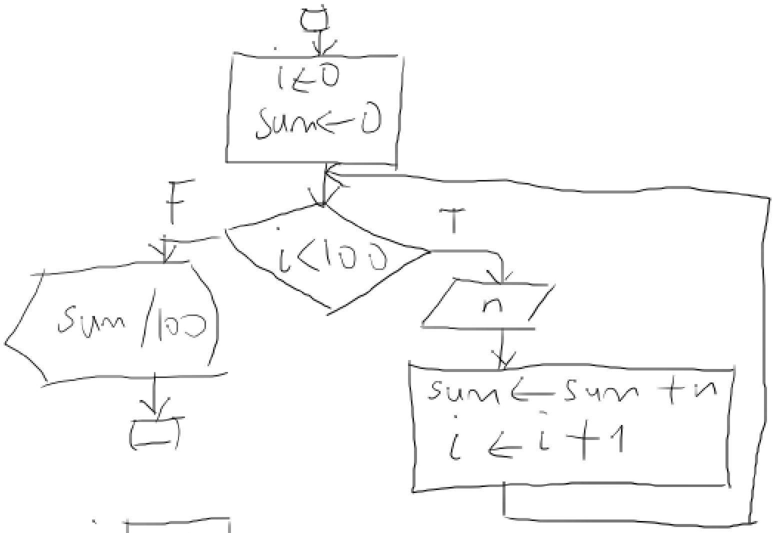
stop point:

```
Read n
res ← 1
while( n > 0 )
      do(   res ← res * n
            n ← n - 1
        )
Print res
```

Ex 8) Read 100 numbers, print the average.

```
i ← 0
sum ← 0
while( i < 100 )
     do(   Read n
           sum ← sum + n
           i ← i + 1
       )
Print sum / 100
```

```
i ← 100
sum ← 0
while( i > 0 )
     do(   Read n
           sum ← sum + n
           i ← i - 1
       )
Print sum / 100
```



i	n	sum
~~0~~	~~5~~	~~5~~
~~1~~	~~8~~	~~13~~
~~2~~	10	~~23~~
3		
~~98~~	~~5~~	2500
~~99~~	10	2510
100		

Ex 8) Read numbers until a negative number arrives. Print the average.

```
count ← 0
sum ← 0
Read n
while( n >= 0 )
      do(   sum ← sum + n
            count ← count + 1
            Read n
        )
if( n = 0 )
      then( Print “No numbers!” )
      else( Print “Average of”, count, “numbers is:”, sum / count )
```