- Previous Lecture:
  - Examples on vectors (1-d arrays)
- Today's Lecture:
  - 2-d array—matrix
- Announcements:
  - Prelim 1 tonight, 7:30-9pm
    - A – G → Goldwin Smith 132
    - H – L → Goldwin Smith G76
    - M – S → Bradfield 101
    - T – Z → Goldwin Smith G64
  - Fall Break: We will post a discussion exercise for next week. On Wednesday section instructors will be in the classrooms as usual. Attendance is optional, but the content of the exercise is not!

A Cost/Inventory Problem

- A company has 3 factories that make 5 different products
- The cost of making a product varies from factory to factory
- The inventory varies from factory to factory

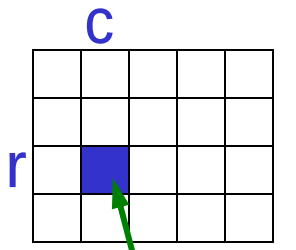
Cost Array

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

The value of `C(i,j)` is what it costs factory i to make product j.

2-d array: **matrix**



- An array is a named collection of like data organized into rows and columns
- A 2-d array is a table, called a *matrix*
- Two *indices* identify the position of a value in a matrix, e.g.,

  `mat(r,c)`

  refers to component in row r, column c of matrix mat
- Array index starts at 1
- Rectangular: all rows have the same #of columns

Creating a matrix

- Built-in functions: ones, zeros, rand
  - E.g., zeros(2,3) gives a 2-by-3 matrix of 0s
- "Build" a matrix using square brackets, [ ], but the dimension must match up:
  - [x y] puts y to the right of x
  - [x; y] puts y below x
  - [4 0 3; 5 1 9] creates the matrix

4	0	3
5	1	9

  - [4 0 3; ones(1,3)] gives

4	0	3
1	1	1

  - [4 0 3; ones(3,1)] doesn't work

Function size returns the dimensions of a matrix

- [nr, nc]= size(M) % nr is #of rows,
  % nc is #of columns

- nr= size(M, 1) % # of rows
- nc= size(M, 2) % # of columns

What will **A** be?

```
A= [1  1]
A= [A'  ones(2,1)]
A= [1  1  1  1;  A  A]
```

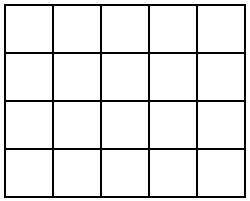
A 3-by-4 matrix

B 4-by-3 matrix

C vector of length 12

D *Error*

Example: minimum value in a matrix



```
function val = minInMatrix(M)
% val is the smallest value in matrix M
```

minInMatrix.m

Pattern for traversing a matrix M

```
[nr, nc] = size(M)
for  r= 1:nr
      % At row r
      for  c= 1:nc
            % At column c (in row r)
            %
            % Do something with M(r,c) …
      end
end
```

A Cost/Inventory Problem

- A company has 3 factories that make 5 different products
- The cost of making a product varies from factory to factory
- The inventory varies from factory to factory

Problems

A customer submits a purchase order that is to be filled by a single factory.

1. How much would it cost a factory to fill the order?
2. Does a factory have enough inventory to fill the order?
3. Among the factories that can fill the order, who can do it most cheaply?

Cost Array

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

The value of `C(i,j)` is what it costs factory i to make product j.

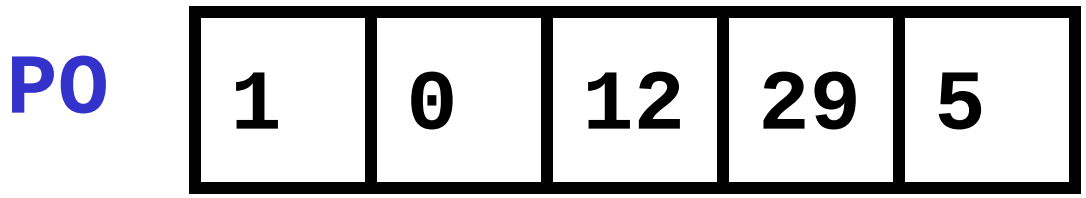
Inventory Array

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

The value of `Inv(i,j)` is the inventory in factory i of product j.

Purchase Order



The value of `PO(j)` is the number of product j's that the customer wants

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5

Cost for factory 1:

1*10 + 0*36 + 12*22 + 29* 15 + 5*62

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5

Cost for factory 1:

```
s = 0;  %Sum of cost
for j=1:5
   s = s + C(1,j)*PO(j)
end
```

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5

Cost for factory 2:

```
s = 0;  %Sum of cost
for j=1:5
   s = s + C(2,j)*PO(j)
end
```

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5

Cost for factory i:

```
s = 0;  %Sum of cost
for j=1:5
   s = s + C(i,j)*PO(j)
end
```

Encapsulate…

```
function  TheBill = iCost(i,C,PO)
% The cost when factory i fills
% the purchase order

nProd = length(PO);
TheBill = 0;
for j=1:nProd
   TheBill = TheBill + C(i,j)*PO(j);
end
```

Finding the Cheapest

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5

1019

930

1040

As computed by **iCost**

Finding the Cheapest

```
iBest = 0;  minBill = inf;
for i=1:nFact
    iBill = iCost(i,C,PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end
```

`inf` – a special value that can be regarded as positive infinity

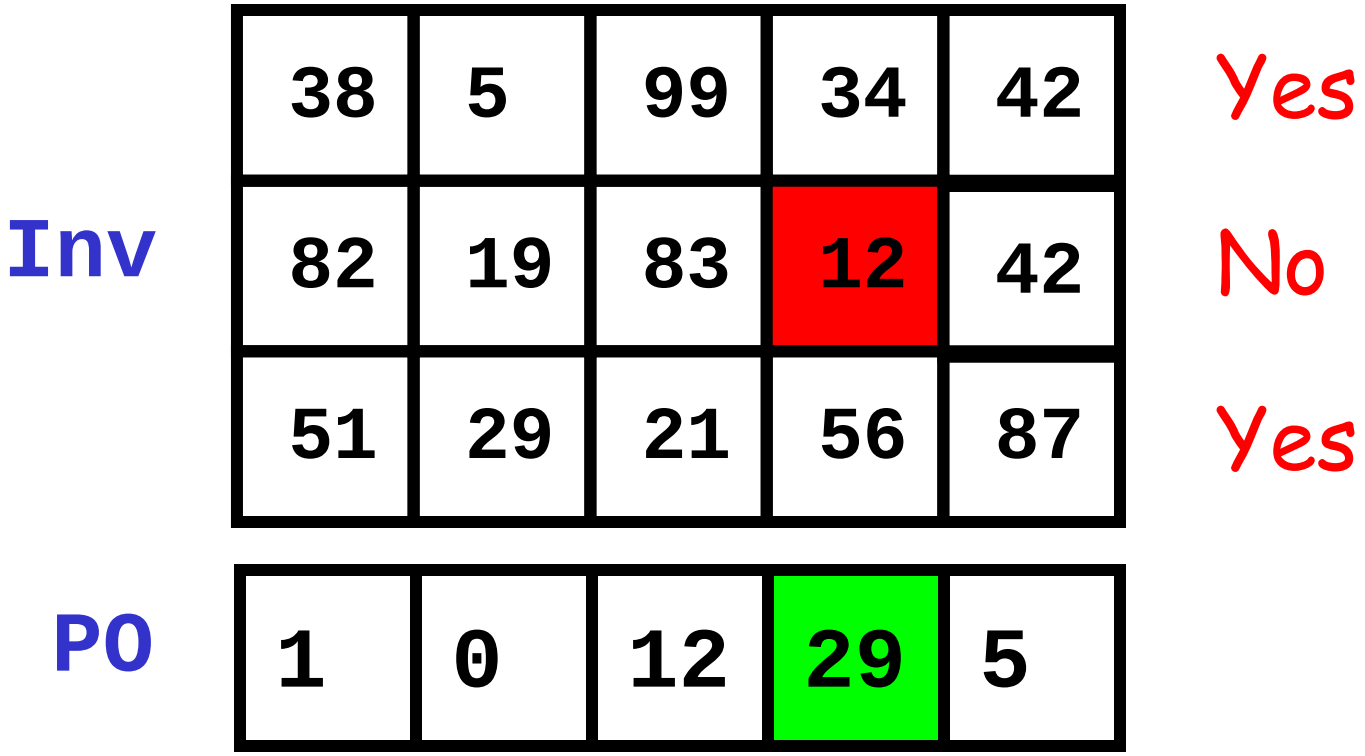
`x = 10/0`	assigns `inf` to `x`
`y = 1+x`	assigns `inf` to `y`
`z = 1/x`	assigns `0` to `z`
`w < inf`	is always true if `w` is numeric

Inventory Considerations

What if a factory lacks the inventory to fill the purchase order?

Such a factory should be excluded from the find-the-cheapest computation.

Who Can Fill the Order?

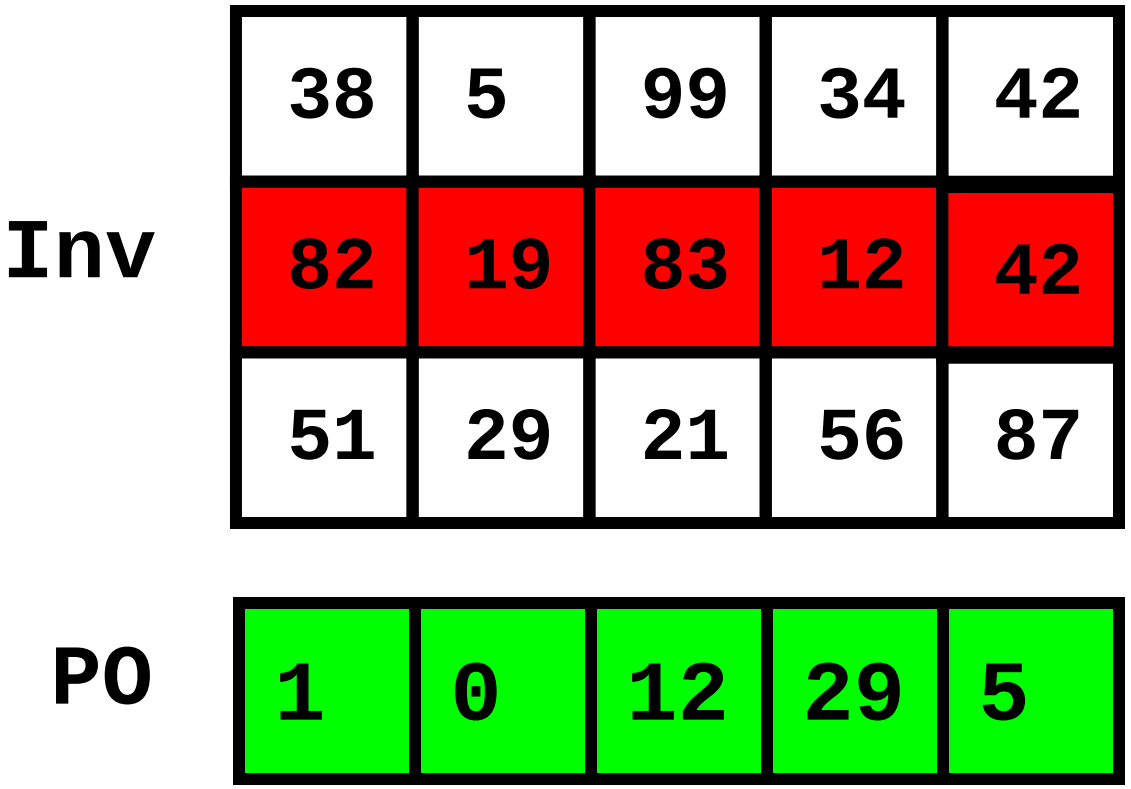


Wanted: A True/False Function

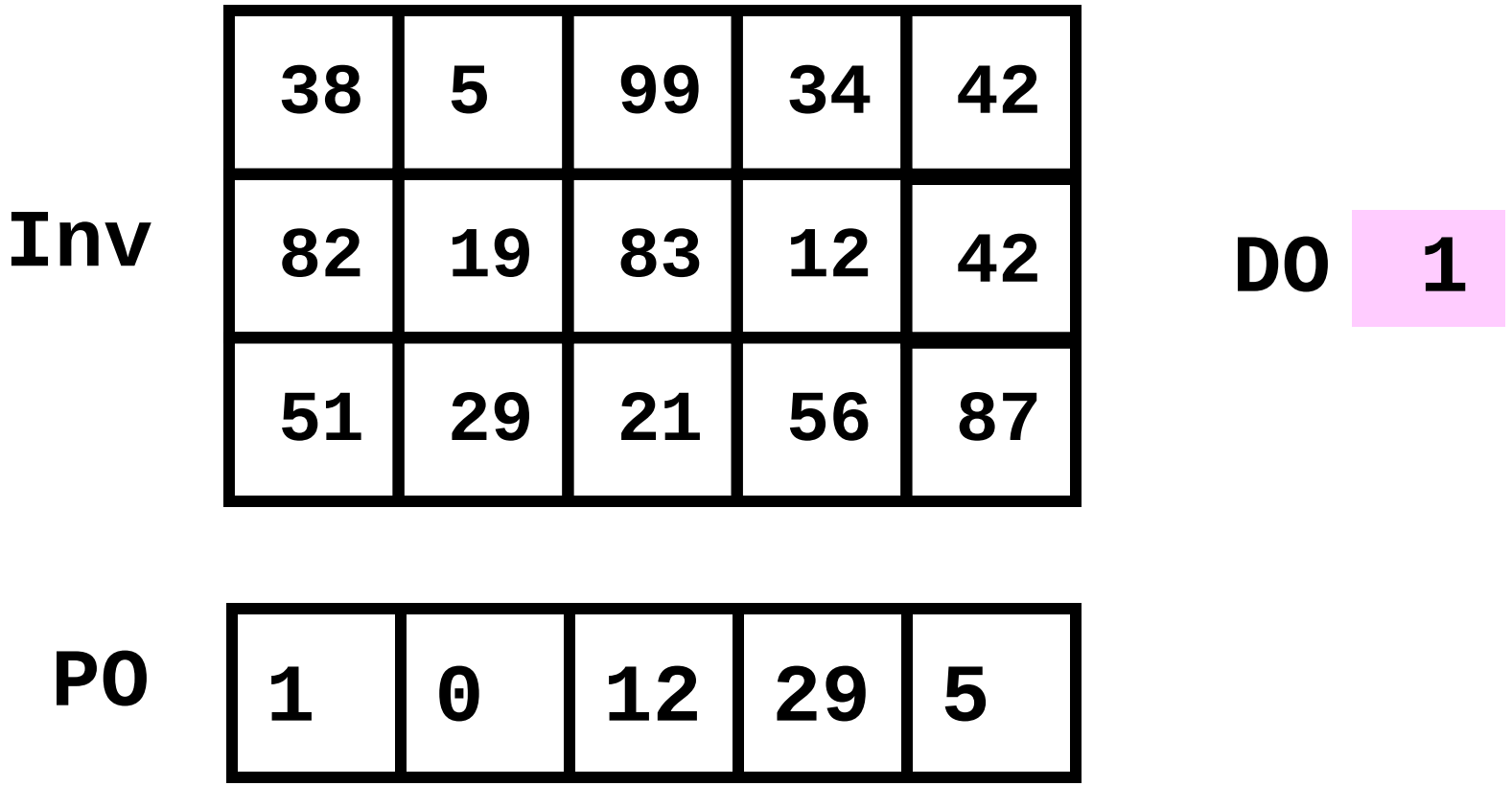
DO is "true" if factory i can fill the order.

DO is "false" if factory i cannot fill the order.

Example: Check inventory of factory 2



Initialization



Still True…

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

DO 1

PO

1	0	12	29	5

```
DO = DO && ( Inv(2,1) >= PO(1) )
```

Still True…

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

DO 1

PO

1	0	12	29	5

```
DO = DO && ( Inv(2,2) >= PO(2) )
```

Still True…

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

DO 1

PO

1	0	12	29	5

```
DO = DO && ( Inv(2,3) >= PO(3) )
```

No Longer True…

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

DO 0

PO

1	0	12	29	5

```
DO = DO && ( Inv(2,4) >= PO(4) )
```

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false

nProd = length(PO);
DO = 1;
for j = 1:nProd
     DO = DO && ( Inv(i,j) >= PO(j) );
end
```

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i,j)>=PO(j)
   j = j+1;
end
DO = ________;
```

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i,j)>=PO(j)
    j = j+1;
end
DO = ________;
```

DO should be true when…

A `j < nProd`

B `j == nProd`

C `j > nProd`

Encapsulate…

```
function  DO = iCanDo(i,Inv,PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j<=nProd && Inv(i,j)>=PO(j)
    j = j+1;
end
DO = j>nProd;
```

Back To Finding the Cheapest

```
iBest = 0; minBill = inf;
for i=1:nFact
    iBill = iCost(i,C,PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end
```

Don't bother with this unless there is sufficient inventory.

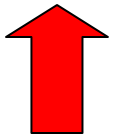
Back To Finding the Cheapest

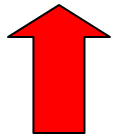
```
iBest = 0; minBill = inf;
for i=1:nFact
    if iCanDo(i,Inv,PO)
        iBill = iCost(i,C,PO);
        if iBill < minBill
        % Found an Improvement
            iBest = i; minBill = iBill;
        end
    end
end
```

Cheapest.m

Finding the Cheapest

	10	36	22	15	62	1019	Yes
C	12	35	20	12	66	930	No
	13	37	21	16	59	1040	Yes
PO	1	0	12	29	5		





As computed by **iCost**

As computed by **iCanDo**