2.6 Reading SAS Data Sets (Self-Study)

Objectives

- Read data from (multiple) SAS data sets to formulate linear programming problems.

173

Reading Data from SAS Data Sets: Example

How can you read the height, weight, and age of students into the arrays **height**, **weight**, and **age**?

SAS Data Set: **opt.class** (only the first four rows)

	Name	Height	Weight	Age
1	Alfred	69	112.5	14
2	Alice	56.5	84	13
3	Barbara	65.3	98	13
4	Carol	62.8	102.5	14

174 ...

The READ DATA statement saves the student names as the elements of the set **STUDENTS**, which must already be declared (as must the parameter arrays **height**, **weight**, and **age**).

Reading Data from SAS Data Sets: Example

How can you read the height, weight, and age of students into the arrays **height**, **weight**, and **age**?

SAS Data Set: **opt.class** (only the first four rows)

	Name	Height	Weight	Age
1	Alfred	69	112.5	14
2	Alice	56.5	84	13
3	Barbara	65.3	98	13
4	Carol	62.8	102.5	14

```
read data opt.class into STUDENTS=[Name]
     height weight age;
```

175

These other data set options might be used when you read or create SAS data sets from PROC OPTMODEL:

FIRSTOBS=	To skip observations at the start of the SAS data set
PW=	To access a password-protected SAS data set
READ=	To assign a password to a SAS data set
RENAME=	To change the name of SAS data set variables*
WHERE=	To read or create a SAS data set that contains only observations meeting certain criteria

* SAS data set variables can be read into parameter arrays with different names in PROC OPTMODEL.

Reading the Students Data Set in PROC OPTMODEL

```
proc optmodel;
   /* declare variables */
   set <str> STUDENTS;
   num height {STUDENTS}, weight {STUDENTS}, age {STUDENTS};

   /* read data from SAS data sets */
   read data opt.class into STUDENTS=[Name] height weight age;

   print height weight age;

quit;
```

Before submitting the program, you must first create a SAS library and assign the name **opt** as the libref.

PROC OPTMODEL Output

The OPTMODEL Procedure

[1]	height	weight	age
Alfred	69.0	112.5	14
Alice	56.5	84.0	13
Barbara	65.3	98.0	13
Carol	62.8	102.5	14
Henry	63.5	102.5	14
James	57.3	83.0	12
Jane	59.8	84.5	12
Janet	62.5	112.5	15
Jeffrey	62.5	84.0	13
John	59.0	99.5	12
Joyce	51.3	50.5	11
Judy	64.3	90.0	14
Louise	56.3	77.0	12
Mary	66.5	112.0	15
Philip	72.0	150.0	16
Robert	64.8	128.0	12
Ronald	67.0	133.0	15
Thomas	57.5	85.0	11
William	66.5	112.0	15

SAS Log

```
NOTE: There were 19 observations read from the data set OPT.CLASS.
```

The OBS data set option is useful for checking the syntax of a READ DATA statement by printing the first few observations.

```
   /* read data from SAS data sets */
   read data opt.class(obs=4) into STUDENTS=[Name] height weight age;
```

PROC OPTMODEL Output

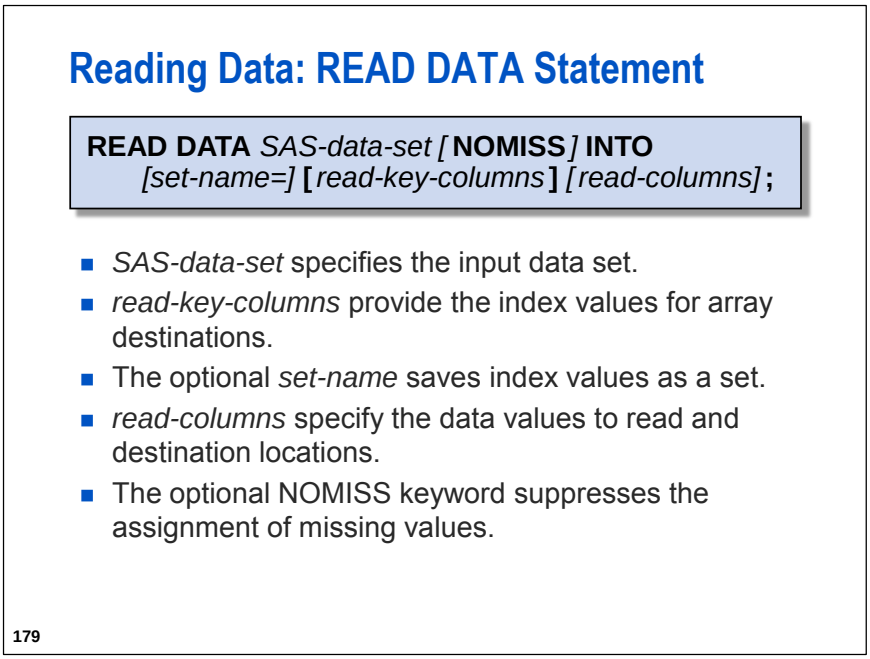
The OPTMODEL Procedure

[1]	height	weight	age
Alfred	69.0	112.5	14
Alice	56.5	84.0	13
Barbara	65.3	98.0	13
Carol	62.8	102.5	14

SAS Log

```
NOTE: There were 4 observations read from the data set OPT.CLASS.
```

End of Demonstration



The syntax for a read column parameter has several forms, including an iterated form.

identifier-expression [/ trim-option]

identifier-expression = *name [/ trim-option]*

identifier-expression = **COL(***name-expression***)** *[/ trim-option]*

- *identifier-expression* specifies the target.
- The name of the input data set variable can be specified with *name* or a COL expression (which is evaluated at read time). Otherwise, the name part of *identifier-expression* is used.
- *trim-option* controls how leading and trailing blanks are treated for strings. The default trim option removes leading and trailing blanks.

{ *index-set* **}** **<** *read-columns* **>**

- This form iterates *read-columns* for each element of *index-set*. Distinct input columns can be specified using dummy parameters from *index-set* in the COL expressions.

Iterated read columns ***cannot*** be nested.

Next, you read the data for the furniture-making problem.

Furniture-Making Problem SAS DATA Steps

```
data resource_data;
   input Resource $ Cost Amount_Available;
   datalines;
labor 14 225
metal 20 117
wood  11 420
run;

data product_data;
   length Item $9;
   input Item $ Selling_Price labor metal wood;
   datalines;
desks      94 2 1 3
chairs     79 1 1 3
bookcases 125 3 1 4
bedframes 109 2 1 4
run;
```

181

Furniture-Making Problem SAS Data Sets

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

182

There are other ways that the same data can be stored in SAS data sets.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

183 ...

These READ DATA statements are examined step by step.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

- Declare index sets. (With no initialization expression, **<str>** is necessary because the default is **<num>**.)

184 ...

The index sets are populated by the READ DATA statements.

Reading the Furniture-Making Data Sets

```
proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;
```

- Declare parameter arrays indexed by the (unpopulated) index sets **PRODUCTS** and **RESOURCES**.

185

Most information in PROC OPTMODEL is stored symbolically and resolved only when necessary.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

read key column
(SAS data set variable name)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

186 ...

The values in the read key column of the SAS data set should be unique. If not, a warning message is generated in the SAS log file. Subsequent observations overwrite earlier observations with the same value in the read key column.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

set name
(OPTMODEL index set)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

187 ...

The set name does not need to be the plural form of the read key column. The former is a PROC OPTMODEL name and the latter is a SAS data set variable name. They can be the same or completely unrelated.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

read columns
(OPTMODEL array name *[=SAS data set variable name]*)

SAS Data Set: **work.resource_data**

	Resource	Cost	Amount_Available
1	labor	14	225
2	metal	20	117
3	wood	11	420

188

In the first case, no SAS data set variable name is supplied, so it is assumed to be **cost** (**Cost**).

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost availability=Amount_Available;
```

The read columns in the READ DATA statement can be seen as an abbreviated version of the read columns in the following equivalent READ DATA statement:

```
set <str> PRODUCTS, RESOURCES;
num cost {RESOURCES}, availability {RESOURCES};

read data resource_data into RESOURCES=[Resource]
   cost[Resource]=Cost
   availability[Resource]=Amount_Available;
```

189

This form of the read columns syntax more closely resembles SAS DATA step syntax.

The name of the read key column functions as a local dummy parameter, which can be used to refer to array locations in the read columns. This is necessary in the READ DATA statement for the product data.

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

read key column (SAS data set variable name)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

190 ...

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

set name (OPTMODEL index set)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

191 ...

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

read column (OPTMODEL array name)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

192 ...

No SAS data set variable name is supplied, so it is assumed to be **selling_price** (**Selling_Price**).

Reading the Furniture-Making Data Sets

```
set <str> PRODUCTS, RESOURCES;
num selling_price {PRODUCTS};
num required {PRODUCTS, RESOURCES};

read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

iterated read column (array destination=COL expression)

SAS Data Set: **work.product_data**

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3
3	bookcases	125	3	1	4
4	bedframes	109	2	1	4

193

Local dummy parameters **r** and **Item** are used to specify array locations. (String functions can be applied to the local dummy parameter in the COL expression to match SAS data set variable names.)

The iterated read column can be replaced by three read columns.

A Closer Look at the Iterated Read Column

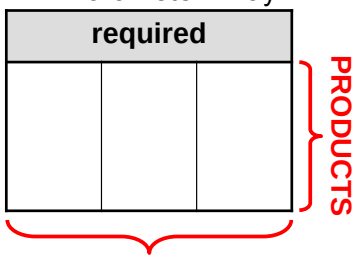
```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>;
```

```
read data product_data into PRODUCTS=[Item]
   required[Item,'labor']=labor
   required[Item,'metal']=metal
   required[Item,'wood']=wood;
```

SAS Data Set (last columns)

	labor	metal	wood
1	2	1	3
2	1	1	3
3	3	1	4
4	2	1	4

2-D Parameter Array



RESOURCES={'labor','metal','wood'}

194

These read columns illustrate the difference between string values and names. (See the COL expression in the iterated read column.)

Space in OPTMODEL: Broad Overview

Declaration statements:

- Types are assigned, so the compiler can check syntax.
- Storage for arrays is allocated one element at a time, when the element is first referenced or assigned.

READ DATA statements:

- These statements populate sets and parameter arrays.

PRINT statements:

- These statements cause storage to be allocated.

SOLVE or EXPAND statements:

- The model (or partial model) is built from index sets and parameter arrays.
- Storage is allocated for suffixed parameter arrays.

195

The following sequence of slides shows how the last READ DATA statement for the furniture-making problem populates the first two rows of the parameter arrays **required** and **selling_price**. The set **RESOURCES** is already populated by the first READ DATA statement.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required	selling_price
2	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

196 ...

For the first observation, the character string in the **Item** column is added to the set **PRODUCTS**. The value in the **labor** column is assigned to a location in the parameter array **required**.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required		selling_price
2	1	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

197 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	

PRODUCTS

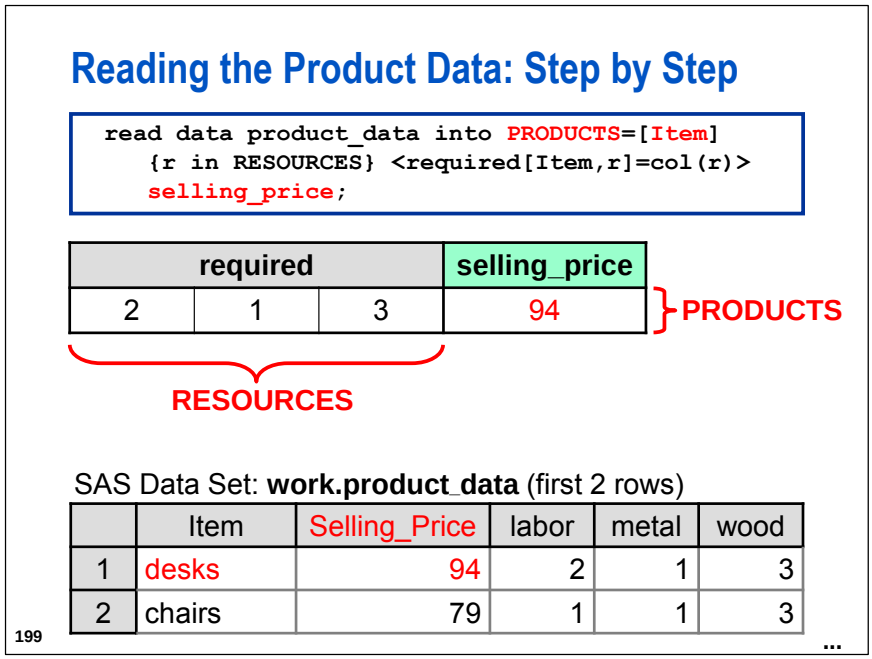
RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

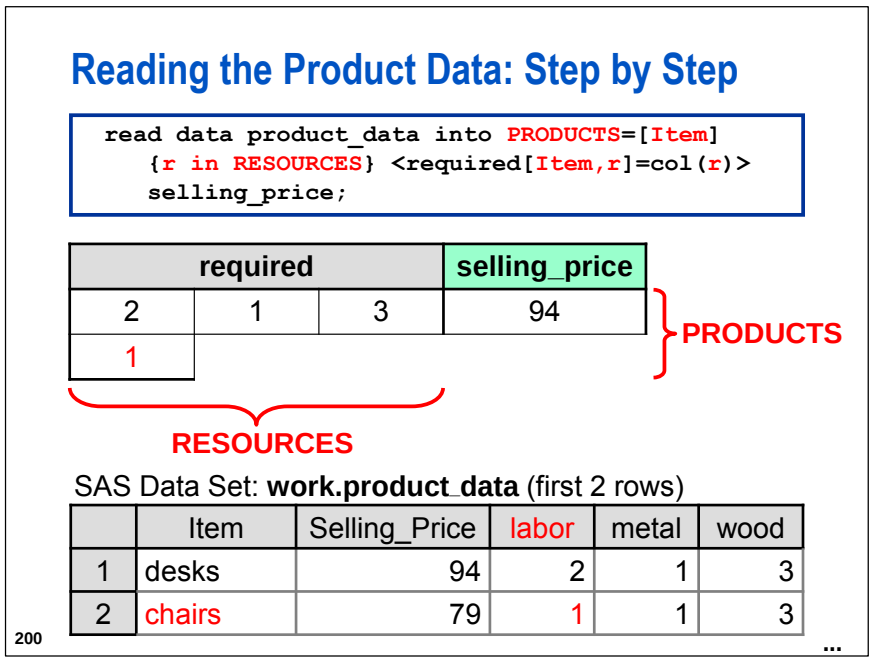
	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

198 ...

After the set **RESOURCES** is iterated, the read column is processed.



The value in the **Selling_Price** column of the SAS data set is assigned to a location in the parameter array **selling_price**.



For the second observation, the character string in the **Item** column is added to the set **PRODUCTS**, and the process continues.

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1		

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

201 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1	3	

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

202 ...

Reading the Product Data: Step by Step

```
read data product_data into PRODUCTS=[Item]
   {r in RESOURCES} <required[Item,r]=col(r)>
   selling_price;
```

required			selling_price
2	1	3	94
1	1	3	79

PRODUCTS

RESOURCES

SAS Data Set: **work.product_data** (first 2 rows)

	Item	Selling_Price	labor	metal	wood
1	desks	94	2	1	3
2	chairs	79	1	1	3

203

The order in which the columns are read from the SAS data set is not important.

Reading SAS Data Sets in PROC OPTMODEL for the Furniture-Making Problem

```
data resource_data;
   input Resource $ Cost Amount_Available;
   datalines;
labor 14 225
metal 20 117
wood  11 420
run;

data product_data;
   length Item $9;
   input Item $ Selling_Price labor metal wood;
   datalines;
desks      94 2 1 3
chairs     79 1 1 3
bookcases 125 3 1 4
bedframes 109 2 1 4
run;

%let budget_limit = 10000;

proc optmodel;
   /* declare sets and parameters */
   set <str> PRODUCTS, RESOURCES;
   num cost {RESOURCES}, availability {RESOURCES};
   num selling_price {PRODUCTS};
   num required {PRODUCTS, RESOURCES};

   /* read data from SAS data sets */
   read data resource_data into RESOURCES=[Resource]
      cost availability=Amount_Available;

   read data product_data into PRODUCTS=[Item]
      {r in RESOURCES} <required[Item,r]=col(r)>
      selling_price;

   /* print parameter arrays */
   print cost dollar. availability;
   print selling_price dollar. required;

   /* declare variables */
   var NumProd {PRODUCTS} >= 0;

   impvar Revenue = sum {p in PRODUCTS}
      selling_price[p] * NumProd[p];

   impvar AmountUsed {r in RESOURCES} =
      sum {p in PRODUCTS} NumProd[p] * required[p,r];
```

```
   impvar TotalCost = sum {r in RESOURCES}
      cost[r] * AmountUsed[r];

   /* declare constraints */
   con Usage {r in RESOURCES}:
      AmountUsed[r] <= availability[r];

   con Budget: TotalCost <= &budget_limit;

   /* declare objective */
   max NetProfit = Revenue - TotalCost;

   solve;

   print NumProd;
   print AmountUsed;

quit;
```

The SAS macro language can be used to input the values of individual parameters, with one macro variable per parameter.

Notice the format applied to the **cost** and **selling_price** parameter arrays in the PRINT statement.

PROC OPTMODEL Output (only the first two PRINT statements)

The OPTMODEL Procedure

[1]	cost	availability
labor	$14	225
metal	$20	117
wood	$11	420

[1]	selling_ price
bedframes	$109
bookcases	$125
chairs	$79
desks	$94

	required			
	bedframes	bookcases	chairs	desks
labor	2	3	1	2
metal	1	1	1	1
wood	4	4	3	3

The problem summary, solution summary, and the output of the last PRINT statements are omitted, because they are identical to the output of the previous furniture-making demonstration program from a previous section.

If the product data is read from the SAS data set below, it is not necessary to read the data from the additional column.

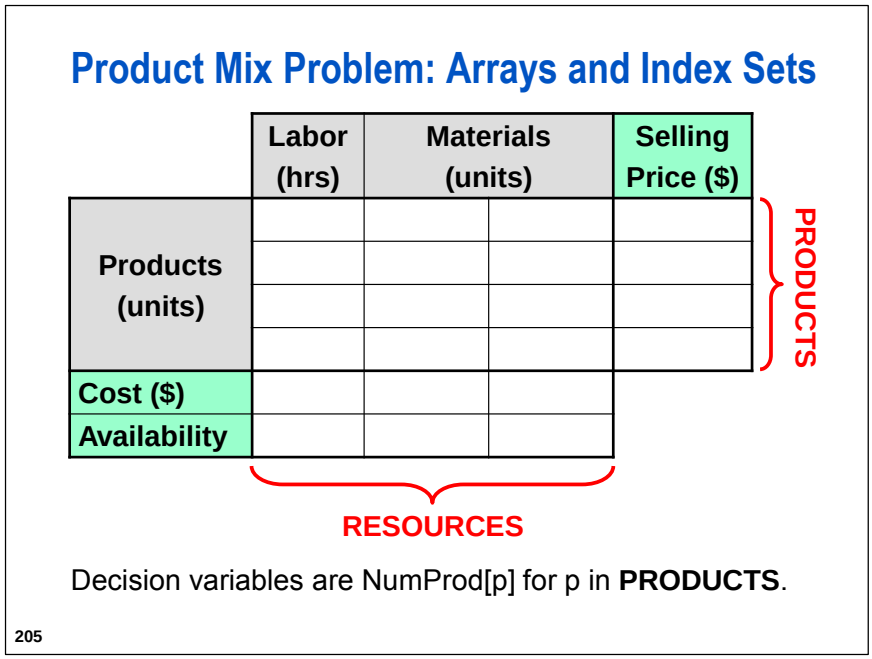
opt.product_data

	Item	Setup_Time	Selling_Price	labor	metal	wood
1	desks	0.3	94	2	1	3
2	chairs	0.6	79	1	1	3
3	bookcases	0.5	125	3	1	4
4	bedframes	0.8	109	2	1	4

Not Used

Notice that, except for the SAS DATA steps, nothing in the PROC OPTMODEL program is specific to the furniture-making problem. In fact, the model is generic and can be used to solve a class of problems called *product mix problems*.

End of Demonstration



These problems might or might not include a budget constraint.