# Chapter 6
# Array Processing

Arthur Li

# Introduction to Array Processing
## Situations for Utilizing Array Processing

Program 6.1:

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

- 6 measurements of SBP for each patient
- The missing values are coded as 999
- Suppose you would like to recode 999 to periods (.)

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

- Each of the IF statements are almost identical
- Only the variable names are different
- Use a DO loop?

# Situations for Utilizing Array Processing

```
data trial2(drop = rannum);
    id = 'M2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2340';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'M1240';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;
run;
```

## ❖RECALL: DO LOOP

```
data trial2 (drop = rannum);
  do id = 'M2390', 'F2390',
          'F2340', 'M1240';
     rannum = ranuni(2);
     if rannum> 0.5 then group = 'D';
     else group ='P';
     output;
  end;
run;
```

Difference:
The values of ID variables

# Situations for Utilizing Array Processing

```
data trial2(drop = rannum);
    id = 'M2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2390';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'F2340';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;

    id = 'M1240';
    rannum = ranuni(2);
    if rannum> 0.5 then group = 'D';
    else group ='P';
    output;
run;
```

- RECALL: DO LOOP

```
data trial2 (drop = rannum);
  do id = 'M2390', 'F2390',
          'F2340', 'M1240';
     rannum = ranuni(2);
     if rannum> 0.5 then group = 'D';
     else group ='P';
     output;
  end;
run;
```

- The loop iterates along a sequence of values
- The index variable holds these values

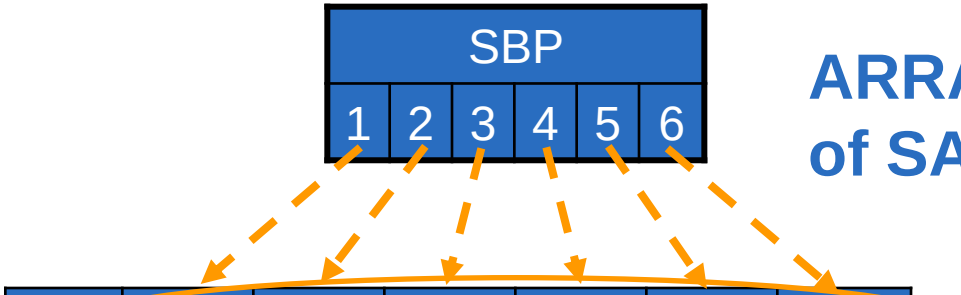
# Situations for Utilizing Array Processing

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

Difference:
Variable names

# Situations for Utilizing Array Processing




| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

**ARRAY: a temporary grouping of SAS variables**

If we can group these variables into a single unit
→
We can loop along these variables

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

Difference:
Variable names

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- does not become part of the output data.
- must be a SAS name.
- cannot be the name of a SAS variable.
- cannot be used in the LABEL, FORMAT, DROP, KEEP, or LENGTH statements

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- The [subscript] component describes the number of array elements.
- The simple form is specifying the dimensional size of the array.

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- $ indicates that the elements in the array are character elements
- $ is not necessary if the elements have been previously defined as character elements

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- If the lengths of array elements have not been previously specified, you can use the length option

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- ARRAY-ELEMENTS are the variables to be included in the array
- Must either be all numeric or characters

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

```
array sbparray [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- You can also provide a range of numbers as [subscript]

```
array sbpary[1990:1993] sbp1990 sbp1991 sbp1992 sbp1993;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- You can use an asterisk (*) as SUBSCRIPT
- You must include ELEMENTS

```
array sbparray [*] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- SUBSCRIPT can be enclosed in parentheses, braces, or brackets

```
array sbparray (6) sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
array sbparray {6} sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
array sbparray [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- If ARRAY-ELEMENTS are not specified, for example:

```
array sbp [6];
```

=

```
array sbp [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- If ARRAY-ELEMENTS are not specified, for example:

```
array sbp [6];
```

=

```
array sbp [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

Situation # 1: sbp1 – sbp6 were previously defined in the DATA step

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- If ARRAY-ELEMENTS are not specified, for example:

```
array sbp [6];
```

=

```
array sbp [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

Situation # 2: if sbp1 – sbp6 were not previously defined in the DATA step, they will be created by the ARRAY statement

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

```
array sbp [6] sbp1 sbp2 sbp3 sbp4 sbp5 sbp6;
```

=

```
array sbp [6] sbp1-sbp6;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- _NUMERIC_ : all numeric variables
- _CHARACTER_ : all character variables
- _ALL_: all the variables; variables must be either all numeric or character

```
array num [*] _numeric_;
```

```
array char [*] _character_;
```

```
array allvar [*] _all_;
```

# Defining and Referencing One-dimensional Arrays

**ARRAY** array-name [subscript] <$><length>
<array-elements> <(initial-value-list)>;

- You can assign initial values to the array elements

```
array num[3] n1 n2 n3 (1 2 3);
```

```
array chr[3] $ ('A', 'B', 'C');
```

# Defining and Referencing One-dimensional Arrays

- Temporary arrays contain temporary data elements
- They are not output to the output dataset
- Using temporary arrays is useful when you want to create an array only for calculation purposes
- When referring to a temporary data element, you refer to it by the ARRAYNAME and its SUBSCRIPT
- You cannot use the asterisk (*) with temporary arrays
- They are always automatically retained
- To create a temporary array, you need to use the keyword _TEMPORARY_

```
array num[3] _temporary_ (1 2 3);
```

# Defining and Referencing One-dimensional Arrays

- To reference an array element:

```
array-name[subscript]
```

- must be closed in ( ), [ ], or { }
- is specified as an integer, a numeric variable, or a SAS expression
- must be within the lower and upper bounds of the DIMENSION of the array

# Defining and Referencing One-dimensional Arrays

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

## ARRAY:

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 sbp2 sbp3
                     sbp4 sbp5 sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

# Defining and Referencing One-dimensional Arrays

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

## ARRAY:

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;

  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

# Defining and Referencing One-dimensional Arrays

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

```
data ex6_1;
  set sbp;
  if sbp1 = 999 then sbp1 = .;
  if sbp2 = 999 then sbp2 = .;
  if sbp3 = 999 then sbp3 = .;
  if sbp4 = 999 then sbp4 = .;
  if sbp5 = 999 then sbp5 = .;
  if sbp6 = 999 then sbp6 = .;
run;
```

## ARRAY:

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;

  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

```
data ex6_2 (drop=i);
  set sbp;
  array sbp [6];
  do i = 1 to 6;
     if sbp [i] = 999 then
        sbp [i] = .;
  end;
run;
```

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

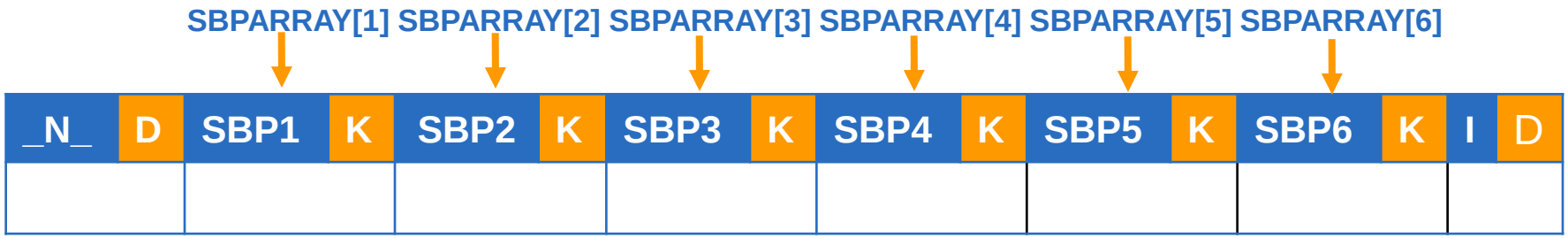
| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | |

- During the compilation phase, PDV is created

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |




- Array name SBPARRAY and references are not included in the PDV
- SBP1 – SBP6, is referenced by the ARRAY reference
- Syntax errors in the ARRAY statement will be detected during the compilation phase

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | . | | . | | . | | . | | . | | . | | . | |

## 1st iteration of the DATA step:

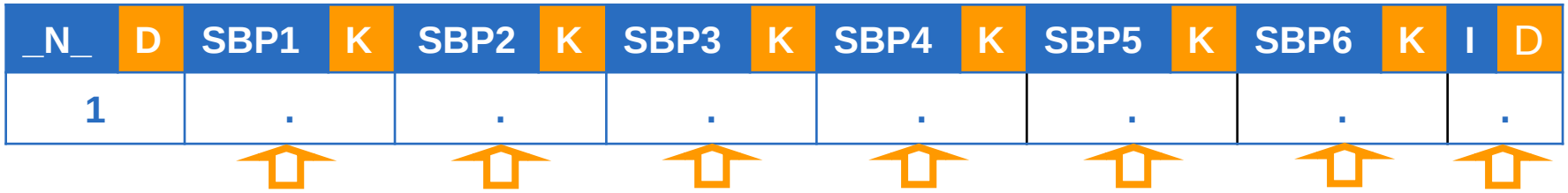
- _N_ ← 1

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]




## 1st iteration of the DATA step:

- The rest of the variables ← *missing*

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | . | |

## 1st iteration of the DATA step:

- SET statement copies the 1st obs. from Sbp to the PDV

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | . | |

## 1st iteration of the DATA step:

- The ARRAY statement is a compile-time only statement

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 1 | |

## 1st iteration of the DATA step:

## 1st iteration of the DO loop:

- I ← 1

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
   if sbparray [i] = 999 then
       sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 1 | |

## 1st iteration of the DATA step:

## 1st iteration of the DO loop:

- SBPARRAY [ i ] → SBPARRAY [1]
- SBPARRAY [1] → SBP1
- Since SBP1 ≠ 999, no execution

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 1 | |

## 1st iteration of the DATA step:

## 1st iteration of the DO loop:

- SAS reaches the end of the DO loop

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 2 | |

## 1st iteration of the DATA step:

## 2nd iteration of the DO loop:

- I ← 2
- Since I ≤ 6, the loop continues

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
  ➔ if sbparray [i] = 999 then
       sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 2 | |

## 1st iteration of the DATA step:

## 2nd iteration of the DO loop:

- SBPARRAY [ i ] → SBPARRAY [2]
- SBPARRAY [2] → SBP2
- Since SBP2 ≠ 999, no execution

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 2 | |

## 1st iteration of the DATA step:

## 2nd iteration of the DO loop:

- SAS reaches the end of the DO loop
- Skip the rest of the iterations

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | 7 | |

## 1st iteration of the DATA step:

- SAS reaches the end of the DATA step
- The implicit OUTPUT executes

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | 141 | | 142 | | 137 | | 117 | | 116 | | 124 | | . | |

## 2nd iteration of the DATA step:

- _N_ ↑ 2
- SBP1 – SBP6 are retained
- I ← *missing*

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Exe  ution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | 999 | | 141 | | 138 | | 119 | | 119 | | 122 | | . | |

## 2nd iteration of the DATA step:

- The SET statement copies the 2nd obs. to the PDV

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | 999 | | 141 | | 138 | | 119 | | 119 | | 122 | | 1 | |

## 2nd iteration of the DATA step:

## 1st iteration of the DO loop:

- I ← 1

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
  → if sbparray [i] = 999 then
       sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | 999 | | 141 | | 138 | | 119 | | 119 | | 122 | | 1 | |

## 2nd iteration of the DATA step:

## 1st iteration of the DO loop:

- SBPARRAY [ i ] → SBPARRAY [1]
- SBPARRAY [1] → SBP1

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
  → if sbparray [i] = 999 then
       sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | . | | 141 | | 138 | | 119 | | 119 | | 122 | | 1 | |

## 2nd iteration of the DATA step:

## 1st iteration of the DO loop:

- SBPARRAY [ i ] → SBPARRAY [1]
- SBPARRAY [1] → SBP1
- Since SBP1 = 999, SBP1 ← *missing*

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | . | | 141 | | 138 | | 119 | | 119 | | 122 | | 1 | |

## 2nd iteration of the DATA step:

## 1st iteration of the DO loop:

- SAS reaches the end of loop
- Skip the rest of the loop

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
  set sbp;
  array sbparray [6] sbp1 - sbp6;
  do i = 1 to 6;
     if sbparray [i] = 999 then
        sbparray [i] = .;
  end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | . | | 141 | | 138 | | 119 | | 119 | | 122 | | 7 | |

## 2nd iteration of the DATA step:

- SAS reaches the end of the DATA step

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |

# Compilation and Execution Phases for Array Processing

```
data ex6_2 (drop=i);
   set sbp;
   array sbparray [6] sbp1 - sbp6;
   do i = 1 to 6;
      if sbparray [i] = 999 then
         sbparray [i] = .;
   end;
run;
```

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | 999 | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | 999 | 139 | 119 | 120 | 999 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |

SBPARRAY[1] SBPARRAY[2] SBPARRAY[3] SBPARRAY[4] SBPARRAY[5] SBPARRAY[6]

| _N_ | D | SBP1 | K | SBP2 | K | SBP3 | K | SBP4 | K | SBP5 | K | SBP6 | K | I | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | | . | | 141 | | 138 | | 119 | | 119 | | 122 | | 7 | |

## 2nd iteration of the DATA step:

- SAS reaches the end of the DATA step
- The implicit OUTPUT executes
- Skip the rest of the iterations

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |

# The DIM, HBOUND, and LBOUND Functions

- The DIM function returns the number of elements in a specified dimension.
- It is convenient when you use _NUMERIC_, _CHARACTER_, _ALL_ as array elements.

**DIM**<n>(array-name)

- If the N value is not specified, the DIM function will return the number of elements in the first dimension of the array.

# The DIM, HBOUND, and LBOUND Functions

- The HBOUND function returns the upper bound in a specified dimension:

**HBOUND**<n>(array-name)

- The LBOUND function returns the lower bound in a specified dimension:

**LBOUND**<n>(array-name)

# The DIM, HBOUND, and LBOUND Functions

## Program 6.3

```
data ex6_3(drop=i);
     set sbp;
     array sbpary[*] _numeric_;
     do i = 1 to dim(sbpary);
          if sbpary[i] = 999 then sbpary[i] = .;
     end;
run;
```

# Using the IN and OF Operator with an Array

- Create a variable, MISS, which is used to indicate whether SBP1 – SBP6 contains missing values.

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 | miss |
|---|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 | 0 |
| 2 | . | 141 | 138 | 119 | 119 | 122 | 1 |
| 3 | 142 | . | 139 | 119 | 120 | . | 1 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 | 0 |

# Using the IN and OF Operator with an Array

- Create a variable, MISS, which is used to indicate whether SBP1 – SBP6 contains missing values.

| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 | miss |
|---|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 | 0 |
| 2 | . | 141 | 138 | 119 | 119 | 122 | 1 |
| 3 | 142 | . | 139 | 119 | 120 | . | 1 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 | 0 |

## Program 6.4

```
data ex6_4;
     set sbp2;
     array sbpary[*] _numeric_;
     miss = . IN sbpary;
run;
```

# Using the IN and OF Operator with an Array

- You can pass an array on to most functions with the OF operator.
- Create two variables: SBP_MIN and SBP_MAX:

Program 6.5

```
data ex6_5;
    set sbp2;
    array sbpary[*] _numeric_;
    sbp_min = min(of sbpary[*]);
    sbp_max = max(of sbpary[*]);
run;

title 'Using the OF operator to create variables SBP_MIN and
SBP_MAX';
proc print data=ex6_5;
run;
```

# Using the IN and OF Operator with an Array

Using the OF operator to create variables SBP_MIN and SBP_MAX

| Obs | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 | sbp_min | sbp_max |
|---|---|---|---|---|---|---|---|---|
| 1 | 141 | 142 | 137 | 117 | 116 | 124 | 116 | 142 |
| 2 | . | 141 | 138 | 119 | 119 | 122 | 119 | 141 |
| 3 | 142 | . | 139 | 119 | 120 | . | 119 | 142 |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 | 118 | 142 |

# Some Array Applications
## Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

# Some Array Applications
## Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

| | above1 | above2 | above3 | above4 | above5 | above6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | . | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | . | 0 | 0 | 0 | . |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |

# Some Array Applications
## Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

| | above1 | above2 | above3 | above4 | above5 | above6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | . | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | . | 0 | 0 | 0 | . |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |

```
data ex6_6 (drop=i);
   set sbp2;
   array sbp[6];
   array above[6];
   array threshhold[6] _temporary_ (140 140 140 120 120 120);
   do i = 1 to 6;
      if (not missing(sbp[i]))
         then above [i] = sbp[i] > threshhold[i];
   end;
run;
```

# Some Array Applications
## Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

| | above1 | above2 | above3 | above4 | above5 | above6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | . | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | . | 0 | 0 | 0 | . |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |

```
data ex6_6 (drop=i);
   set sbp2;
   array sbp[6];
   array above[6];
   array threshhold[6] _temporary_ (140 140 140 120 120 120);
   do i = 1 to 6;
      if (not missing(sbp[i]))
         then above [i] = sbp[i] > threshhold[i];
   end;
run;
```

**Used to group the existing variables: sbp1 – sbp6**

# Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

| | above1 | above2 | above3 | above4 | above5 | above6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | . | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | . | 0 | 0 | 0 | . |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |

```
data ex6_6 (drop=i);
   set sbp2;
   array sbp[6];
   array above[6];
   array threshhold[6] _temporary_ (140 140 140 120 120 120);
   do i = 1 to 6;
      if (not missing(sbp[i]))
         then above [i] = sbp[i] > threshhold[i];
   end;
run;
```

**Used to create variables: above1 – above6**

# Creating a Group of Variables by Using Arrays

| | Pre-treatment | | | Post-treatment | | |
|---|---|---|---|---|---|---|
| | sbp1 | sbp2 | sbp3 | sbp4 | sbp5 | sbp6 |
| 1 | 141 | 142 | 137 | 117 | 116 | 124 |
| 2 | . | 141 | 138 | 119 | 119 | 122 |
| 3 | 142 | . | 139 | 119 | 120 | . |
| 4 | 136 | 140 | 142 | 118 | 121 | 123 |
| MEAN SBP: | | 140 | | | 120 | |

| | above1 | above2 | above3 | above4 | above5 | above6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 2 | . | 1 | 0 | 0 | 0 | 1 |
| 3 | 1 | . | 0 | 0 | 0 | . |
| 4 | 0 | 0 | 1 | 0 | 1 | 1 |

```
data ex6_6 (drop=i);
   set sbp2;
   array sbp[6];
   array above[6];
   array threshhold[6] _temporary_ (140 140 140 120 120 120);
   do i = 1 to 6;
      if (not missing(sbp[i]))
         then above [i] = sbp[i] > threshhold[i];
   end;
run;
```

**The temporary array is for comparison purposes**

# Calculating Products of Multiple Variables

Approach:

1. Create an array: num[4]
2. Treat missing value as 1
3. Set result = num[1]
   Loop: i = 2 to 4
   result = result * num[i]
   End Loop

*Product:*

| | num1 | num2 | num3 | num4 |
|---|---|---|---|---|
| 1 | 4 | . | 2 | 3 |
| 2 | . | 2 | 3 | 1 |

# Calculating Products of Multiple Variables

Approach:

1. Create an array: num[4]
2. Treat missing value as 1
3. Set result = num[1]
   Loop: i = 2 to 4
   result = result * num[i]
   End Loop

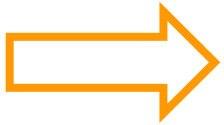
*Product:*

| | num1 | num2 | num3 | num4 |
|---|---|---|---|---|
| 1 | 4 | . | 2 | 3 |
| 2 | . | 2 | 3 | 1 |

```
data ex6_7(drop=i);
     set product;
     array num[4];
     if missing(num[1]) then result = 1;
     else result = num[1];
     do i = 2 to 4;
        if not missing(num[i])
           then result =result*num[i];
     end;
run;
```

# Restructuring Data Sets Using One-dimensional Arrays

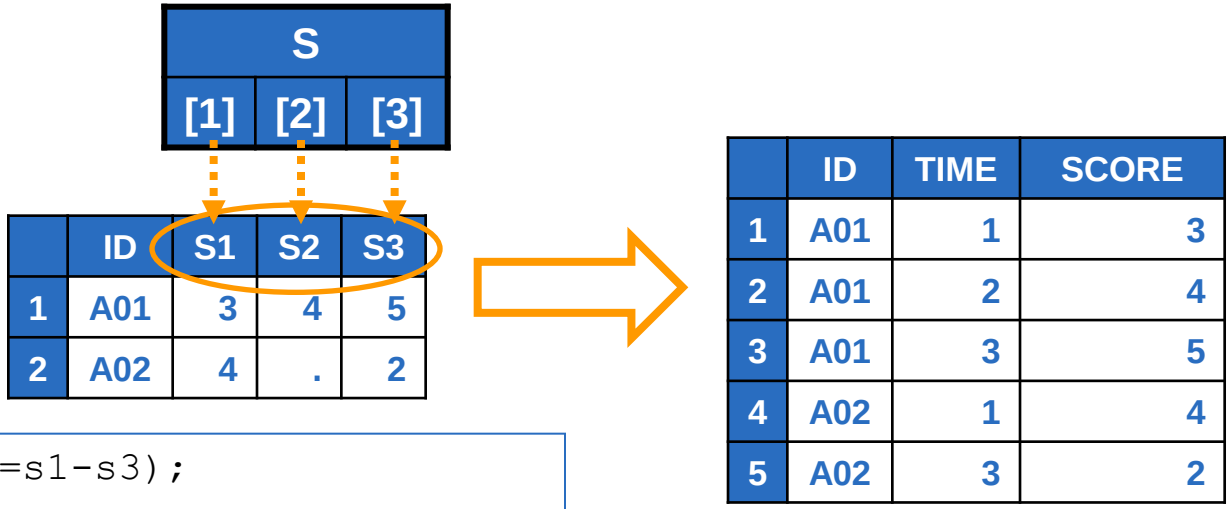
| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |



| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |

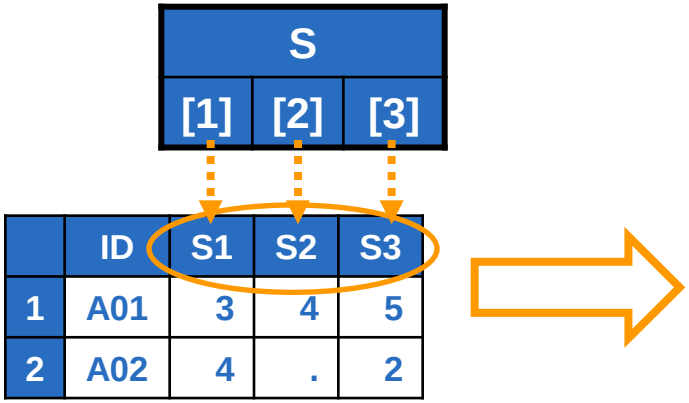
```
data ex6_8(drop=s1-s3);
  set wide;

  time = 1;
  score = s1;
  if not missing(score) then output;

  time = 2;
  score = s2;
  if not missing(score) then output;

  time = 3;
  score = s3;
  if not missing(score) then output;

run;
```

# Restructuring Data Sets Using One-dimensional Arrays




```
data ex6_8(drop=s1-s3);
  set wide;
  array s[3];
  time = 1;
  score = S[1];
  if not missing(score) then output;

  time = 2;
  score = S[2];
  if not missing(score) then output;

  time = 3;
  score = S[3];
  if not missing(score) then output;

run;
```

# Restructuring Data Sets Using One-dimensional Arrays



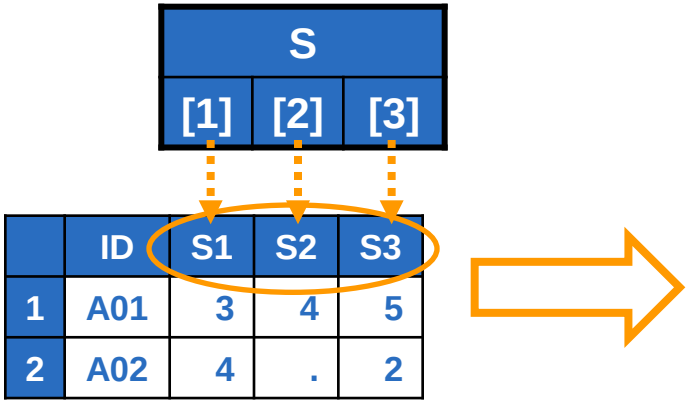

| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |

```
data ex6_8(drop=s1-s3);
  set wide;
  array s[3];
  time = 1;
  score = S[1];
  if not missing(score) then output;
  time = 2;
  score = S[2];
  if not missing(score) then output;
  time = 3;
  score = S[3];
  if not missing(score) then output;
run;
```

```
do time = 1 to 3;
  score = s[time];
  if not missing(score)
    then output;
end;
```

# Restructuring Data Sets Using One-dimensional Arrays




| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

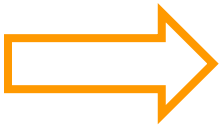
| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |

```
data ex6_8(drop=s1-s3);
  set wide;
  array s[3];
  time = 1;
  score = S[1];
  if not missing(score) then output;
  time = 2;
  score = S[2];
  if not missing(score) then output;
  time = 3;
  score = S[3];
  if not missing(score) then output;
run;
```

```
data ex6_8(drop=s1-s3);
  set wide;
  array s[3];
  do time = 1 to 3;
    score = s[time];
    if not missing(score)
      then output;
  end;
run;
```

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |

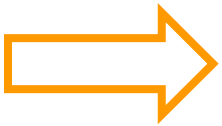


| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
    set long;
    by id;

    retain s1 - s3;

    if first.id then do;
        s1 = .;  s2 = .; s3 = .;
    end;

    if time = 1 then  s1 = score;
    else if time = 2 then  s2 = score;
    else  s3 = score;

    if last.id;
run;
```

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |

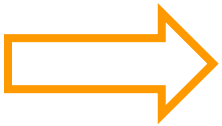


| | | S | | |
|---|---|---|---|---|
| | | [1] | [2] | [3] |
| | ID | S1 | S2 | S3 |
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
    set long;
    by id;
    array s[3];
    retain s;

    if first.id then do;
       S[1] = .; S[2] = .;S[3] = .;
    end;

    if time = 1 then S[1] = score;
    else if time = 2 then S[2] = score;
    else S[3] = score;

    if last.id;
run;
```

# Restructuring Data Sets Using One-dimensional Arrays

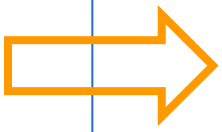
| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |



| | S | | |
|---|---|---|---|
| | [1] | [2] | [3] |

| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

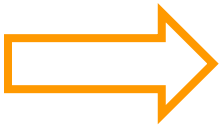
```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .; S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```



```
if first.id then do;
  do i = 1 to 3;
    s[i] = .;
  end;
end;
```

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |



S[1] S[2] S[3]

| | ID | S1 | S2 | S3 |
|---|---|---|---|---|
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .; S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```

| _N_ | D | FIRST.ID | D | LAST.ID | D |
|---|---|---|---|---|---|
| | | | | | |

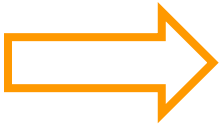
| ID | K | TIME | D | SCORE | D |
|---|---|---|---|---|---|
| | | | | | |

| S1 | K | S2 | K | S3 | K |
|---|---|---|---|---|---|
| | | | | | |

S[1] S[2] S[3]

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |



| | | S | | |
|---|---|---|---|---|
| | | [1] | [2] | [3] |
| | ID | S1 | S2 | S3 |
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .; S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```

| _N_ | D | FIRST.ID | D | LAST.ID | D |
|---|---|---|---|---|---|
| 1 | | 1 | | 0 | |

| ID | K | TIME | D | SCORE | D |
|---|---|---|---|---|---|
| A01 | | 1 | | 3 | |

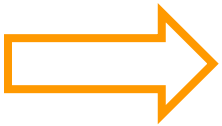
| S1 | K | S2 | K | S3 | K |
|---|---|---|---|---|---|
| 3 | | . | | . | |

S[1] S[2] S[3]

S[TIME]

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |



| | | S | | |
|---|---|---|---|---|
| | | [1] | [2] | [3] |
| | ID | S1 | S2 | S3 |
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .; S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```

| _N_ | D | FIRST.ID | D | LAST.ID | D |
|---|---|---|---|---|---|
| 2 | | 0 | | 0 | |

| ID | K | TIME | D | SCORE | D |
|---|---|---|---|---|---|
| A01 | | 2 | | 4 | |

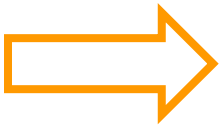
| S1 | K | S2 | K | S3 | K |
|---|---|---|---|---|---|
| 3 | | 4 | | . | |

S[1] S[2] S[3]

S[TIME]

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |



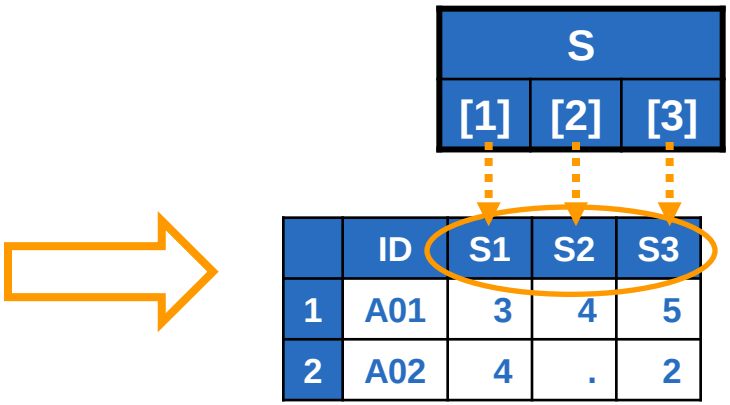
| | | S | | |
|---|---|---|---|---|
| | | [1] | [2] | [3] |
| | ID | S1 | S2 | S3 |
| 1 | A01 | 3 | 4 | 5 |
| 2 | A02 | 4 | . | 2 |

```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .;S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```

```
s[time] = score;
```

# Restructuring Data Sets Using One-dimensional Arrays

| | ID | TIME | SCORE |
|---|---|---|---|
| 1 | A01 | 1 | 3 |
| 2 | A01 | 2 | 4 |
| 3 | A01 | 3 | 5 |
| 4 | A02 | 1 | 4 |
| 5 | A02 | 3 | 2 |




```
data ex6_9(drop=time score);
     set long;
     by id;
     array s[3];
     retain s;

     if first.id then do;
        S[1] = .; S[2] = .;S[3] = .;
     end;

     if time = 1 then S[1] = score;
     else if time = 2 then S[2] = score;
     else S[3] = score;

     if last.id;
run;
```

```
data ex6_9(drop = time score i);
     set long;
     by id;
     array s[3];
     retain s;
     if first.id then do;
       do i = 1 to 3;
         s[i] = .;
       end;
     end;

     s[time] = score;

     if last.id;
  run;
```