# SAS Lesson 09

## Input Data

The **orion.biz_list** data set is extracted from the accounting system and contains the names of Orion Star's U.S. suppliers, charities, and consultants.

Partial Listing of **orion.biz_list**

```
Acct_
Code      Name

AEK3      ANGELA E. KEARNEY
AQI2      AQUAMISSIONS INTERNATIONAL
ATS1      A TEAM SPORTS
CBO3      CLAIRE B. OWENS
CCI2      CANCER CURES, INC.
CNI2      CONSERVE NATURE, INC.
CS1       CAROLINA SPORTS
```

# Input Data – Details

**`Acct_Code`** is a character variable defined as length 6. Its last digit represents the type of organization: **`1`** denotes a supplier, **`2`** a charity, and **`3`** a consultant.

The other characters in the **`Acct_Code`** variable represent the ID for the organization, so the **`ID`** value can have as many as five characters.

Example:

| **`Acct_Code`** <br> **`$6`** | **`ID`** <br> **`$5`** |
|---|---|
| **`AQI2`** | **`AQI`** |

- 2 denotes a charity.
- AQI is the ID.

# Create the List of Charities – Step 1

This program uses the SUBSTR and LENGTH functions to create the **charities** data set.

The LENGTH function is *nested,* or used as an argument to the SUBSTR function.

```
data charities;
  length ID $ 5;
  set orion.biz_list;
  if substr(Acct_Code,length(Acct_Code),1)='2';
  ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Partially stepping through the execution for the first charity observation shows how the functions transform the data.

## Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Read the first charity observation.

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| | AQI2 | AQUAMISSIONS INTERNATIONAL |

## Execution: Step 1

4

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| | AQI2 | AQUAMISSIONS INTERNATIONAL |

# Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| | AQI2 | AQUAMISSIONS INTERNATIONAL |

# Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

True

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| | AQI2 | AQUAMISSIONS INTERNATIONAL |

## Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

3

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| | AQI2 | AQUAMISSIONS INTERNATIONAL |

## Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

**PDV**

| ID | Acct_Code | Name |
|---|---|---|
| $ 5 | $ 6 | $ 30 |
|  | AQI2 | AQUAMISSIONS INTERNATIONAL |

## Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| AQI | AQI2 | AQUAMISSIONS INTERNATIONAL |

## Execution: Step 1

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
run;
```

Implicit OUTPUT;
Implicit RETURN;

**PDV**

| ID<br>$ 5 | Acct_Code<br>$ 6 | Name<br>$ 30 |
|---|---|---|
| AQI | AQI2 | AQUAMISSIONS INTERNATIONAL |

# Create the List of Charities – Step 1 Complete

Listing of **`charities`**

```
          Acct_
  ID      Code       Name

 AQI      AQI2       AQUAMISSIONS INTERNATIONAL
 CCI      CCI2       CANCER CURES, INC.
 CNI      CNI2       CONSERVE NATURE, INC.
 CS       CS2        CHILD SURVIVORS
 CU       CU2        CUIDADORES LTD.
 DAI      DAI2       DISASTER ASSIST, INC.
 ES       ES2        EARTHSALVORS
 FFC      FFC2       FARMING FOR COMMUNITIES
 MI       MI2        MITLEID INTERNATIONAL
 SBA      SBA2       SAVE THE BABY ANIMALS
 V2       V22        VOX VICTIMAS
 YYCR     YYCR2      YES, YOU CAN RECYCLE
```

Step 2 is to transform the values in **`Name`** to a mix of uppercase and lowercase.

## The PROPCASE Function

The PROPCASE function converts all words in an argument to *proper case*, in which the first letter is uppercase and the remaining letters are lowercase.

General form for the PROPCASE function:

```
NewVar=PROPCASE(argument <,delimiter(s)>);
```

| | |
|---|---|
| *argument* | can be a character constant, variable, or expression. |
| *delimiter(s)* | delimiters are characters which separate words. If omitted, the default delimiters are the blank, /, - , ( , ., and tab characters. |
| *NewVar* | If *NewVar* is a new variable, it is created with the same length as *argument*. |

# The PROPCASE Function

Example:

```
Name = 'SURF&LINK SPORTS';
Pname = propcase(Name);
Pname2 = propcase(Name,' &');
```

**PDV**

| Name<br>$ 16 | Pname<br>$ 16 |
|---|---|
| SURF&LINK SPORTS | Surf&link Sports |

| Pname2<br>$ 16 |
|---|
| Surf&Link Sports |

# Quiz

This PDV shows the current value of **Name**:

| Name |
|---|
| HEATH*BARR*LITTLE EQUIPMENT SALES |

Write an assignment statement that converts the value of **Name** to this:

| Name |
|---|
| Heath*Barr*Little Equipment Sales |

# Quiz – Correct Answer

This PDV shows the current value of **Name**:

| Name |
|---|
| HEATH*BARR*LITTLE EQUIPMENT SALES |

Write an assignment statement that will convert the value of **Name** to this:

| Name |
|---|
| Heath*Barr*Little Equipment Sales |

```
Name = propcase(Name,' *');
```

The second argument to the PROPCASE function must list all the characters to use as delimiters. In this example, the space and * both need to be listed.

## Create the List of Charities – Step 2

Adding an assignment statement to convert **Name** to proper case completes the **charities** data set.

```
data charities;
   length ID $ 5;
   set orion.biz_list;
   if substr(Acct_Code,length(Acct_Code),1)='2';
   ID=substr(Acct_Code,1,length(Acct_Code)-1);
   Name = propcase(Name);
run;
```

## Create the List of Charities – Complete

Listing of **`charities`**

```
            Acct_
  ID        Code      Name

 AQI        AQI2      Aquamissions International
 CCI        CCI2      Cancer Cures, Inc.
 CNI        CNI2      Conserve Nature, Inc.
 CS         CS2       Child Survivors
 CU         CU2       Cuidadores Ltd.
 DAI        DAI2      Disaster Assist, Inc.
 ES         ES2       Earthsalvors
 FFC        FFC2      Farming For Communities
 MI         MI2       Mitleid International
 SBA        SBA2      Save The Baby Animals
 V2         V22       Vox Victimas
 YYCR       YYCR2     Yes, You Can Recycle
```

# Other Useful Character Functions

| Function | Purpose |
|---|---|
| RIGHT(*string*) | right-aligns a character expression. |
| LEFT(*string*) | left-aligns a character expression. |
| UPCASE(*string*) | converts all letters in an argument to uppercase. |
| LOWCASE(*string*) | converts all letters in an argument to lowercase. |
| CHAR(*string,position*) | returns a single character from a specified *position* in a character *string*. |

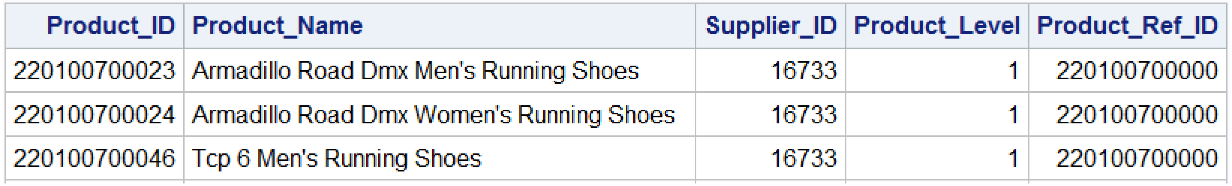
remove leading or trailing blanks.

substr but only 1 character.

# Quiz

Find the syntax error in the code below. Product_Name has a length of 45.

Partial listing of product_list:



| Product_ID | Product_Name | Supplier_ID | Product_Level | Product_Ref_ID |
|---|---|---|---|---|
| 220100700023 | Armadillo Road Dmx Men's Running Shoes | 16733 | 1 | 220100700000 |
| 220100700024 | Armadillo Road Dmx Women's Running Shoes | 16733 | 1 | 220100700000 |
| 220100700046 | Tcp 6 Men's Running Shoes | 16733 | 1 | 220100700000 |

```
data shoes;
   set orion.product_list;
   if substr(right(Product_Name,33,13))=
      'Running Shoes';
run;
```

## Quiz – Correct Answer

Misplaced parentheses are some of the most common syntax errors with functions.

Correctly placed

Corrected program:

```
data shoes;
  set orion.product_list;
  if substr(right(Product_Name),33,13)=
      'Running Shoes';
run;
```

# Business Scenario – Create Mailing List Data

The **orion.contacts** data set contains the contact information for each charity's representative.

Partial Listing of **orion.contacts**

```
ID     Title       Name            Address1              Address2

AQI     Ms.     Farr,Sue       15 Harvey Rd.       Macon, GA  31298
CCI     Dr.     Cox,Kay B.     163 McNeil Pl.      Kern, CA  93280
CNI     Mr.     Mason,Ron      442 Glen Ave.       Miami, FL  33054
CS      Ms.     Ruth,G. H.     2491 Brady St.      Munger, MI  48747
```

**Address1** and **Address2** are in the correct form to use for a mailing address, but the **Title** and **Name** variables need to be combined into a new variable, **FullName**.

# Business Scenario – Desired Output

Create a new data set, **`labels`**, that is suitable for creating mailing labels.

Partial Listing of **`labels`**

```
ID         FullName              Address1             Address2

AQI    Ms. Sue Farr         15 Harvey Rd.       Macon, GA  31298
CCI    Dr. Kay B. Cox       163 McNeil Pl.      Kern, CA  93280
CNI    Mr. Ron Mason        442 Glen Ave.       Miami, FL  33054
CS     Ms. G. H. Ruth       2491 Brady St.      Munger, MI  48747
```

# Create Mailing List Data

Partial Listing of **`orion.contacts`**

```
Title     Name

Ms.       Farr,Sue
Dr.       Cox,Kay B.
Mr.       Mason,Ron
Ms.       Ruth,G. H.
Prof.     Florentino, Helen-Ashe H
Ms.       Van Allsburg, Jan F.
Mr.       Laff, Stanley X.
Mr.       Rizen, George Q.
Dr.       Mitchell, Marc J.
Ms.       Mills, Dorothy E.
Dr.       Webb, Jonathan W.
Mr.       Keenan, Maynard J.
```

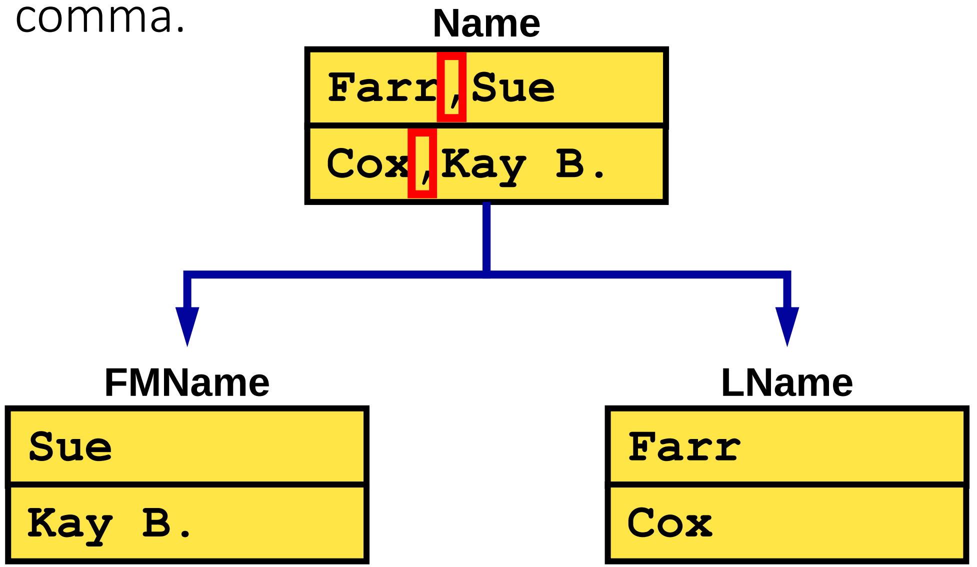
Two steps need to be accomplished:

**Step 1:** Separate the last name from the first and middle names.

**Step 2:** Combine the title, the first and middle names, and the last name.

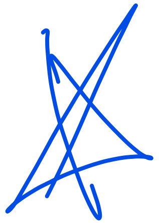
# Separating Data Elements

The first step in creating the mailing list is to separate the contact's name into two parts based on the position of the comma.



Would the SUBSTR function be appropriate for this? ✓ can be done but not best choice.

## The SCAN Function



The SCAN function returns the *n*th word of a character value.

General form of the SCAN function:

*NewVar*=SCAN(*string*,*n*<,*charlist*>);

| | |
|---|---|
| *string* | can be a character constant, variable, or expression. |
| *n* | specifies the *n*th word to extract from *string*. |
| *charlist* | lists the character(s) that delimit words. If omitted, the default delimiters are as follows: |

| | |
|---|---|
| ASCII (PC, UNIX) | blank . < ( + \| & ! $ * ) ; - / , % ^ |
| EBCDIC (z/OS) | blank . < ( + \| & ! $ * ) ; - / , % \| ¢ ¬ |

## The SCAN Function – Details

When you use the SCAN function,

- a missing value is returned if there are fewer than *n* words in the string
- if *n* is negative, the SCAN function selects the word in the character string starting from the end of string
- the length of a new created variable is the length of the first argument.

✎ A good practice is to explicitly define the length of any created variable with a LENGTH statement.

*continued...*

## The SCAN Function – Details

When you use the SCAN function,

- delimiters before the first word have no effect
- any character or set of characters can serve as delimiters
- two or more contiguous delimiters are treated as a single delimiter
- modifiers can be used as a fourth argument to change the default behavior.

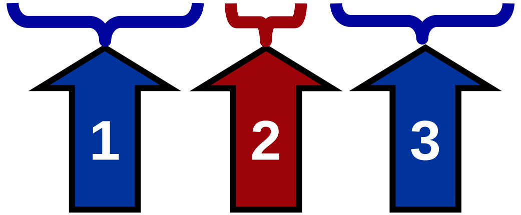
# The SCAN Function – Example

Extract the second word of **`Phrase`**.

```
Second=scan(Phrase,2,' ');
```

**PDV**

| Phrase<br>$ 21 | Second<br>$ 21 |
|---|---|
| software and services | and |



# Quiz

Consider this PDV and assignment statement:

```
Second=scan(Phrase,2,',');
```

**PDV**

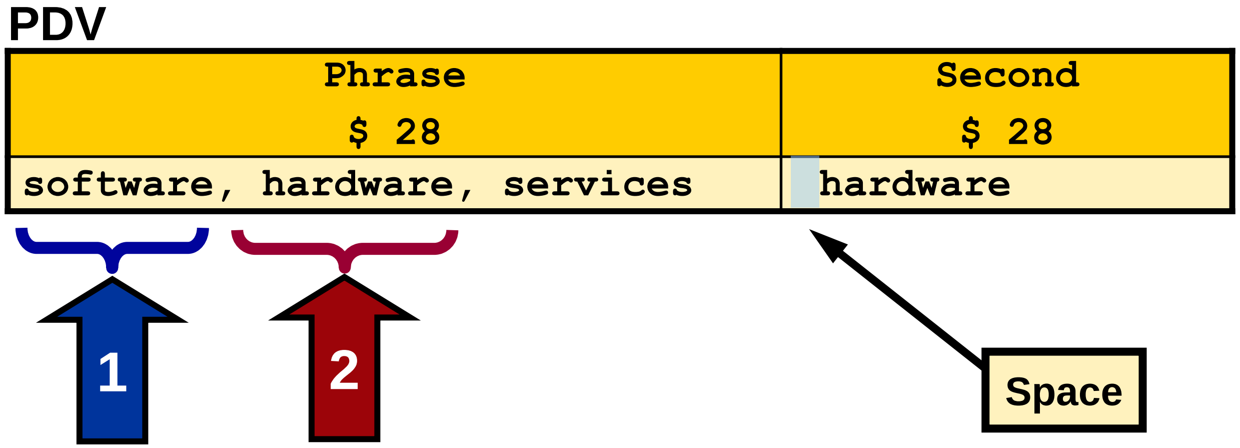
| Phrase<br>$ 28 | Second<br>$ 28 |
|---|---|
| software, hardware, services | |

What value will be stored in **Second**?

# Quiz – Correct Answer

Consider this PDV and assignment statement:

```
Second=scan(Phrase,2,',');
```



PDV

| Phrase<br>$ 28 | Second<br>$ 28 |
|---|---|
| software, hardware, services | ␣hardware |

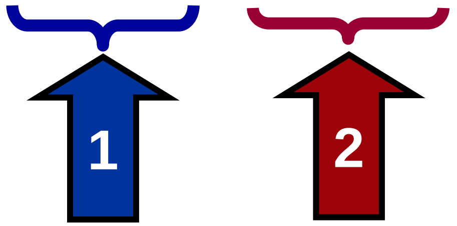
# The SCAN Function – Example

Extract the second word of **`Phrase`** without the leading space.

```
Second=scan(Phrase,2,', ');
```

**PDV**

| Phrase | Second |
|---|---|
| $ 28 | $ 28 |
| software, hardware, services | hardware |



# Multiple Choice Poll

What expression completes the assignment statement to correctly extract 2007 from the **`Text`** variable?

a. **`scan(Text,-1);`**

b. **`scan(Text,6)`**;

c. **`scan(Text,6,', ');`**

[✓ d.] All of the above would work.

```
data Scan_Quiz;
   Text = "New Year's Day, January 1st, 2007";
   Year =             ?             ;
run;
```

# Multiple Choice Poll – Correct Answer

What expression completes the assignment statement to correctly extract 2007 from the **`Text`** variable?

a. **`scan(Text,-1);`**

b. **`scan(Text,6)`**;

c. **`scan(Text,6,', ');`**

[✓ d.] All of the above would work.

## Create Mailing List Data

Using the SCAN function gives an easy way to separate the names for the mailing list.

```
data labels;
   set orion.contacts;
   length FMName LName $ 15;
   FMName = scan(Name,2,',');
   LName = scan(Name,1,',');
run;
```

# Create Mailing List Data

```
proc print data=labels noobs;
   var ID Name Title FMName LName;
run;
```

Partial PROC PRINT Output

```
ID          Name          Title     FMName     LName

AQI     Farr,Sue           Ms.      Sue        Farr
CCI     Cox,Kay B.         Dr.      Kay B.     Cox
CNI     Mason,Ron          Mr.      Ron        Mason
CS      Ruth,G. H.         Ms.      G. H.      Ruth
```

The next step is to join the values of **`Title`**, **`FMName`**, and **`LName`** into another variable.

## The CATX Function

The CATX function joins or *concatenates* character strings.

General form of the CATX function: → leading and trailing blanks are removed,

```
NewVar = CATX(separator, string-1, … ,string-n)
```

| | |
|---|---|
| *separator* | Is a character string that is inserted between the concatenated *string-1,…,string-n* arguments. |
| *string-1, …,string-n* | can be a character constant, variable, or expression. Leading and trailing blanks are removed from each argument. |

The size of the created variable, *NewVar,* is 200 bytes if it is not previously defined with a LENGTH statement. ☆

# The CATX Function – Example

Combine **FMName** and **LName** to create **FullName**.

```
FullName=catx(' ',FMName,LName);
```

**PDV**

| FMName<br>$ 15 | LName<br>$ 15 | FullName<br>$ 200 |
|---|---|---|
| Sue | Farr | Sue Farr |

## Other CAT Functions

There are three other CAT functions that concatenate character strings.

| Function | Details |
|---|---|
| CAT(*string-1, … ,string-n*) | does not remove leading or trailing blanks from the arguments before concatenating them. |
| CATS(*string-1, … ,string-n*) | removes leading and trailing blanks from the arguments. |
| CATT(*string-1, … ,string-n*) | removes trailing blanks from the arguments. |

CATX removes leading & trailing blanks.

## Create Mailing List Data – Finished Program

Adding an assignment statement with the CATX function completes the program.

```
data labels;
   set orion.contacts;
   length FullName $ 35 FMName LName $ 15;
   FMName = scan(Name,2,',');
   LName = scan(Name,1,',');
   FullName = catx(' ',Title,FMName,LName);
run;
```

# Create Mailing List Data – Finished Program

```
proc print data=labels noobs;
   var ID FullName Address1 Address2;
run;
```

Partial PROC PRINT Output

```
ID         FullName            Address1            Address2

AQI    Ms. Sue Farr       15 Harvey Rd.       Macon, GA  31298
CCI    Dr. Kay B. Cox     163 McNeil Pl.      Kern, CA  93280
CNI    Mr. Ron Mason      442 Glen Ave.       Miami, FL  33054
CS     Ms. G. H. Ruth     2491 Brady St.      Munger, MI  48747
```

# Concatenation Operator

The *concatenation operator* is another way to join character strings.

General form of the concatenation operator:

```
NewVar=string1 !! string2;
```

Example:

```
Phone = '('!!area!!') '!!Number;
```

**PDV**

| Area<br>$ 3 | Number<br>$ 8 | Phone<br>$ 14 |
|---|---|---|
| 919 | 531-0000 | (919) 531-0000 |

The operator can also be written as two vertical bars (||) or two broken vertical bars (¦¦).

## Business Scenario: Data Clean Up

The Internet Sales Group accidentally used the wrong data files for the Orion Star Catalog Web site. They corrected the problem as soon as it was noticed, but some orders were created with data errors in them.

**`orion.clean_up`** has sample observations showing the problems.

# Business Scenario: Data Clean Up

Listing of **`orion.clean_up`**

```
Product_ID          Product                       Order_ID

21 02 002 00003     Sunfit Trunks, Blue           1231986335
21 02 002 00003     Luci Knit Mittens, Red        1232003930
21 02 002 00004     Luci Knit mittens, Blue       1232007693
21 02 002 00004     Sunfit Trunks, aqua           1232007700
21 02 002 00005     Sunfit Trunks, Yellow         1232087464
21 02 002 00005     Lucky Knit Mittens, Black     1232092527
```

- The **`Product_ID`** for mittens should have `5` instead of a `2` for the third group of numbers.
- `Luci` is a typo; the correct word is `Lucky`.
- **`Product_ID`** values should have no internal spaces.
- All words in the **`Product`** value should start with a capital letter.

## Business Scenario – Desired Output

The **`correct`** data set shows what the data should be.

Listing of **`correct`**

```
Product_ID          Product                         Order_ID

210200200003        Sunfit Trunks, Blue             1231986335
210200500003        Lucky Knit Mittens, Red         1232003930
210200500004        Lucky Knit Mittens, Blue        1232007693
210200200004        Sunfit Trunks, Aqua             1232007700
210200200005        Sunfit Trunks, Yellow           1232087464
210200500005        Lucky Knit Mittens, Black       1232092527
```

# Data Clean Up – Step 1

The first step in creating the **`correct`** data set is to do the following:

- Find the observations with `Mittens` as part of the **`Product`** value.
- Change the middle characters of the **`Product_ID`** values for those observations.

The FIND and SUBSTR functions are useful for this.

## The FIND Function

The FIND function searches a target string for a specified substring.

General form of the FIND function:

```
Position = FIND(string,substring<,modifiers,startpos>);
```

The FIND function returns a numeric value that is

- the starting position of the first occurrence of *substring* within *string*, if *substring* is found
- 0, if *substring* is not found.

*continued...*

# The FIND Function

The FIND function searches a target *string* for a specified *substring*.

General form of the FIND function:

```
Position = FIND(string,substring<,modifiers,startpos>);
```

*Modifiers* can be

- I to indicate a case-insensitive search
- T to indicate to ignore trailing blanks in the string and substring values.

*startpos* indicates where in the *string* to start searching for the *substring*.

## The INDEX Function

The INDEX function searches a target *string* for a specified *substring*.

INDEX is a predecessor to the FIND function and performs the same operation without the optional arguments.

General form of the INDEX function:

```
Position = INDEX(string, substring);
```

# The FIND Function – Example

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 |
|---|
| |

What value will SAS assign to **Pos1**?

# The FIND Function – Example

2

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 |
|---|
| 2 |

## The FIND Function – Example

20

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 | Pos2<br>N 8 |
|---|---|
| 2 | 20 |

# Quiz

Complete the PDV for the values for **Pos3** and **Pos4**.

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 | Pos2<br>N 8 | Pos3<br>N 8 | Pos4<br>N 8 |
|---|---|---|---|
| 2 | 20 | | |

# Quiz – Correct Answer

Complete the PDV for the values for **Pos3** and **Pos4**.

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 | Pos2<br>N 8 | Pos3<br>N 8 | Pos4<br>N 8 |
|---|---|---|---|
| 2 | 20 | 0 | 2 |

# The FIND Function – Example

21

```
data find;
   Text='AUSTRALIA, DENMARK, US';
   Pos1=find(Text,'US');
   Pos2=find(Text,' US');
   Pos3=find(Text,'us');
   Pos4=find(Text,'us','I');
   Pos5=find(Text,'us','I',10);
run;
```

**PDV**

| Pos1<br>N 8 | Pos2<br>N 8 | Pos3<br>N 8 | Pos4<br>N 8 | Pos5<br>N 8 |
|---|---|---|---|---|
| 2 | 20 | 0 | 2 | 21 |

## The SUBSTR Function (Left Side)

This form of the SUBSTR function (left side of assignment statement) replaces characters in a character variable.

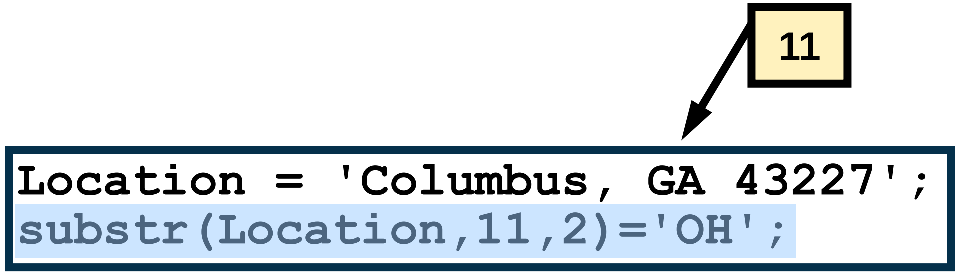
General form of the SUBSTR function (left side):

```
SUBSTR(string,start<,length>)=value;
```

| | |
|---|---|
| *string* | specifies a character variable. |
| *start* | specifies the starting position to replace characters with the *value.* |
| *length* | specifies the number of characters to replace in *string*. If omitted, all characters from the *start* position to the end of the *string* are replaced.<br>The length value cannot be larger than the remaining length of *string* (including trailing blanks) after *start*. |

## The SUBSTR Function (Left Side) – Example

Replace two characters starting at position 11.



```
Location = 'Columbus, GA 43227';
substr(Location,11,2)='OH';
```

**PDV**

| Location<br>$ 18 |
|---|
| Columbus, OH 43227 |

# Data Clean Up – Step 1

Use the SUBSTR and FIND functions to change incorrect product IDs for mittens.

```
data correct;
   set orion.clean_up;
   if find(Product,'Mittens','I')>0 then do;
      substr(Product_ID,9,1) = '5';
   end;
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 1

PROC PRINT Output

```
Product_ID          Product                        Order_ID

21 02 002 00003     Sunfit Trunks, Blue            1231986335
21 02 005 00003     Luci Knit Mittens, Red         1232003930
21 02 005 00004     Luci Knit mittens, blue        1232007693
21 02 002 00004     Sunfit Trunks, aqua            1232007700
21 02 002 00005     Sunfit Trunks, Yellow          1232087464
21 02 005 00005     Lucky Knit Mittens, Black      1232092527
```

The next step is to change the error `Luci` to `Lucky`.

The TRANWRD function is the best way to do this kind of change.

# The TRANWRD Function

The TRANWRD function replaces or removes all occurrences of a given word (or a pattern of characters) within a character string.

General form for the TRANWRD function:

```
NewVar=TRANWRD(source,target,replacement);
```

| | |
|---|---|
| *source* | specifies the source string that you want to change. |
| *target* | specifies the string searched for in *source*. |
| *replacement* | specifies the string that replaces *target*. |

## The TRANWRD Function – Details

General form for the TRANWRD function:

```
NewVar=TRANWRD(source,target,replacement);
```

These details apply when you use the TRANWRD function:

- The TRANWRD function does not remove trailing blanks from *target* or *replacement*.
- If *NewVar* was not previously defined, it is given a length of 200.
- If the target string is not found in the source, then no replacement occurs.

# Data Clean Up – Step 2

Use the TRANWRD function to replace all occurrences of `Luci` with `Lucky`.

```
data correct;
   set orion.clean_up;
   if find(Product,'Mittens','I') > 0 then do;
      substr(Product_ID,9,1) = '5';
      Product=Tranwrd(Product,'Luci ','Lucky ');
   end;
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 2

PROC PRINT Output

```
Product_ID         Product                          Order_ID

21 02 002 00003    Sunfit Trunks, Blue             1231986335
21 02 005 00003    Lucky Knit Mittens, Red         1232003930
21 02 005 00004    Lucky Knit mittens, blue        1232007693
21 02 002 00004    Sunfit Trunks, aqua             1232007700
21 02 002 00005    Sunfit Trunks, Yellow           1232087464
21 02 005 00005    Lucky Knit Mittens, Black       1232092527
```

For step 3, removing the embedded blanks from **`Product_ID`** is easy with the COMPRESS function.

## The COMPRESS Function

The COMPRESS function removes the characters listed in the *chars* argument from the *source*.

General form for the COMPRESS function:

```
NewVar=COMPRESS(source<,chars> <,modifiers>);
```

Modifiers control whether specified characters are kept or removed. They can also add specific types of characters to the list as a group. See Prep Guide for full list.

If no optional arguments are specified, the COMPRESS function removes all blanks from the *source*.

## The COMPRESS Function

```
ID ='20 01-005 024';
New_ID1=compress(ID);
New_ID2=compress(ID,'-');
New_ID3=compress(ID,' -');
```

**PDV**

| ID<br>$ 13 | New_ID1<br>$ 13 |
|---|---|
| 20 01-005 024 | 2001-005024 |

| New_ID2<br>$ 13 | New_ID3<br>$ 13 |
|---|---|
| 20 01005 024 | 2001005024 |

## Other Functions That Remove Blanks

| Function | Purpose |
|---|---|
| TRIM(*string*) | removes trailing blanks from a character string. |
| STRIP(*string*) | removes all leading and trailing blanks from a character string. |
| COMPBL(*string*) | removes multiple blanks from a character string by translating each occurrence of two or more consecutive blanks into a single blank. |

# Data Clean Up – Step 3

Use the COMPRESS and PROPCASE functions to eliminate blanks from **Product_ID** and ensure the proper case for **Product**.

```
data correct;
   set orion.clean_up;
   if find(Product,'Mittens','I') > 0 then do;
      substr(Product_ID,9,1) = '5';
      Product=tranwrd(Product,'Luci ','Lucky ');
   end;
   Product_ID = compress(Product_ID);
   Product = propcase(Product);
run;

proc print data=correct noobs;
run;
```

# Data Clean Up – Step 3

PROC PRINT Output

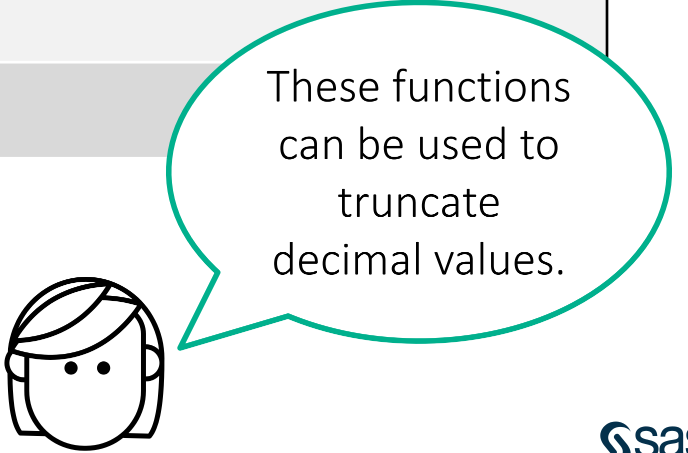
```
Product_ID          Product                            Order_ID

210200200003        Sunfit Trunks, Blue              1231986335
210200500003        Lucky Knit Mittens, Red          1232003930
210200500004        Lucky Knit Mittens, Blue         1232007693
210200200004        Sunfit Trunks, Aqua              1232007700
210200200005        Sunfit Trunks, Yellow            1232087464
210200500005        Lucky Knit Mittens, Black        1232092527
```

## Manipulating Data with Functions

Using Numeric Functions

# Changing Numeric Precision

| Function | What it does |
|---|---|
| ROUND(*number*) | Returns a value rounded to nearest multiple |
| CEIL(*number*) | Returns the smallest integer that is greater than or equal to the argument |
| FLOOR(*number*) | Returns the largest integer that is less than or equal to the argument |
| INT(*number*) | Returns the integer value |



## The ROUND Function

The ROUND function returns a value rounded to the nearest multiple of the round-off unit.

General form of the ROUND function:

```
NewVar=ROUND(argument<,round-off-unit>);
```

| | |
|---|---|
| *argument* | is a number or numeric expression. |
| *round-off-unit* | is numeric and positive. If *round-off-unit* is not provided, *argument* is rounded to the nearest integer. |

## The ROUND Function – Example

```
data truncate;
   NewVar1=round(12.12);
   NewVar2=round(42.65,.1);
   NewVar3=round(-6.478);
   NewVar4=round(96.47,10);
run;
```

**PDV**

| NewVar1<br>N 8 | NewVar2<br>N 8 | NewVar3<br>N 8 | NewVar4<br>N 8 |
|---|---|---|---|
| 12 | 42.7 | -6 | 100 |

## The ROUND Function – Example

```
data truncate;
   NewVar5=round(12.69,.25);
   NewVar6=round(42.65,.5);
run;
```

Round to the nearest multiple of .25

**PDV**

| NewVar5<br>N 8 | NewVar6<br>N 8 |
|---|---|
| 12.75 | . |

...

## The ROUND Function – Example

```
data truncate;
   NewVar5=round(12.69,.25);
   NewVar6=round(42.65,.5);
run;
```

Round to the nearest multiple of .5

**PDV**

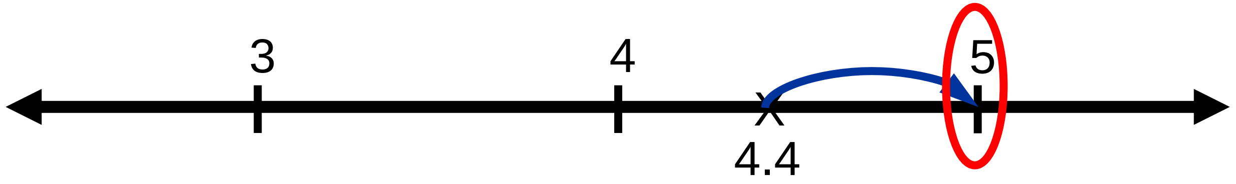
| NewVar5<br>N 8 | NewVar6<br>N 8 |
|---|---|
| 12.75 | 42.5 |

## The CEIL Function

The CEIL function returns the smallest integer greater than or equal to the argument.

General form of the CEIL function:

```
NewVar=CEIL(argument);
```



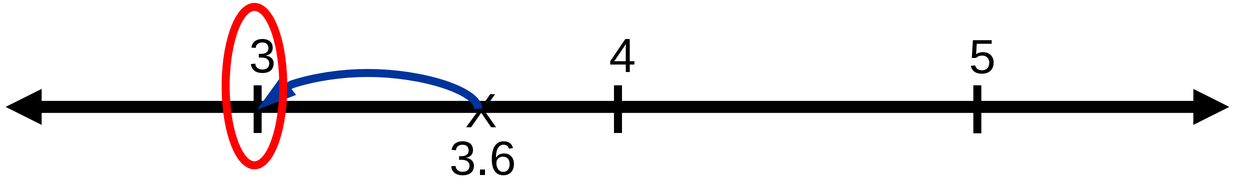
```
x=ceil(4.4);
```

# The FLOOR Function

The FLOOR function returns the greatest integer less than or equal to the argument.

General form of the FLOOR function:

*NewVar*=FLOOR(*argument*);



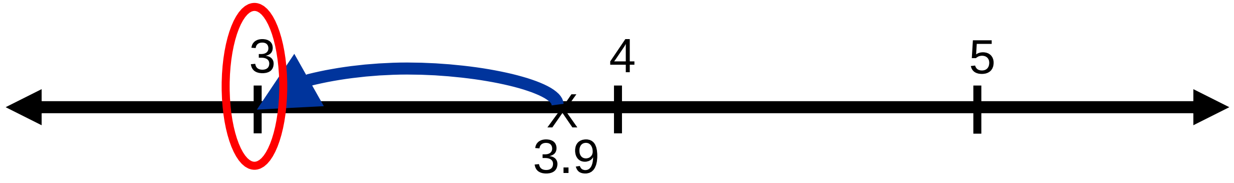
```
y=floor(3.6);
```

# The INT Function

The INT function returns the integer portion of the argument.

General form of the INT function:

```
NewVar=INT(argument);
```



```
z=int(3.9);
```

## Truncation Functions – Example

```
data truncate;
   Var1=6.478;
   CeilVar1=ceil(Var1);
   FloorVar1=floor(Var1);
   IntVar1=int(Var1);
run;
```

**PDV**

| Var1 | CeilVar1 | FloorVar1 | IntVar1 |
|---|---|---|---|
| 6.478 | 7 | 6 | 6 |

# Setup for the Poll

In this program, the values returned from the FLOOR and INT functions are the same.

```
data truncate;
   Var1=6.478;
   CeilVar1=ceil(Var1);
   FloorVar1=floor(Var1);
   IntVar1=int(Var1);
run;
```

**PDV**

| Var1 | CeilVar1 | FloorVar1 | IntVar1 |
|---|---|---|---|
| 6.478 | 7 | 6 | 6 |

## Poll

Given the same value as an argument, do the INT and the FLOOR functions always return the same result?

❍ Yes

❍ No

## Poll – Correct Answer

Given the same value as an argument, do the INT and the FLOOR functions always return the same result?

- Yes
- [✓ No]

The INT and the FLOOR functions give different results if the argument value is negative.

# Truncation Functions

Compare the values from the CEIL, FLOOR, and INT functions with a negative argument.

```
data truncate;
   Var1=-6.478;
   CeilVar1=ceil(Var1);
   FloorVar1=floor(Var1);
   IntVar1=int(Var1);
run;
```

**PDV**

| Var1 | CeilVar1 | FloorVar1 | IntVar1 |
|---|---|---|---|
| -6.478 | -6 | -7 | -6 |

# Numeric Functions

| Functions |
|---|
| SUM (*num1, num2, ...*) |
| MEAN (*num1, num2, ...*) |
| MEDIAN (*num1, num2, ...*) |
| RANGE (*num1, num2, ...*) |
| MIN (*num1, num2, ...*) |
| MAX (*num1, num2, ...*) |
| N (*num1, num2, ...*) |
| NMISS\|CMISS (*num1, num2, ...*) |

Count of non-missing arguments

Count of missing arguments

# Descriptive Statistics Functions

These functions all share the same general syntax:

```
function-name(argument-1,argument-2,…,argument-n)
```

- *argument-1* through *argument-n* are numeric. (except CMISS)
- An argument can be a variable list, which is preceded by OF.
- The non-counting functions ignore missing values in their arguments.

# Descriptive Statistics Functions

```
data descript;
   Var1=12;
   Var2=.;
   Var3=7;
   Var4=5;
   SumVars=sum(Var1,Var2,Var3,Var4);
   AvgVars=mean(of Var1-Var4);
   MissVars=cmiss(of Var1-Var4);
run;
```

**PDV**

| Var1 | Var2 | Var3 | Var4 |
|---|---|---|---|
| 12 | . | 7 | 5 |

| SumVars | AvgVars | MissVars |
|---|---|---|
| 24 | . | . |

# Descriptive Statistics Functions

```
data descript;
   Var1=12;
   Var2=.;
   Var3=7;
   Var4=5;
   SumVars=sum(Var1,Var2,Var3,Var4);
   AvgVars=mean(of Var1-Var4);
   MissVars=cmiss(of Var1-Var4);
run;
```

**PDV**

| Var1 | Var2 | Var3 | Var4 |
|---|---|---|---|
| 12 | . | 7 | 5 |

| SumVars | AvgVars | MissVars |
|---|---|---|
| 24 | 8 | . |

# Descriptive Statistics Functions

```
data descript;
   Var1=12;
   Var2=.;
   Var3=7;
   Var4=5;
   SumVars=sum(Var1,Var2,Var3,Var4);
   AvgVars=mean(of Var1-Var4);
   MissVars=cmiss(of Var1-Var4);
run;
```

**PDV**

| Var1 | Var2 | Var3 | Var4 |
|---:|---:|---:|---:|
| 12 | . | 7 | 5 |

| SumVars | AvgVars | MissVars |
|---:|---:|---:|
| 24 | 8 | 1 |

# Multiple Choice Poll

**`Ord1`**, **`Ord2`**, and **`Ord3`** are variables that contain the sale amounts of the last three orders from a customer. Which of the following expressions can calculate the total of the two largest orders?

a. **`sum(max(of Ord1-Ord3),max(of Ord1-Ord3))`**

[✓ b.] **`sum(of Ord1-Ord3)-min(of Ord1-Ord3)`**

c. **`max(of Ord1-Ord3) + min(of Ord1-Ord3)`**

d. None of the above

# Multiple Choice Poll – Correct Answer

**`Ord1`**, **`Ord2`**, and **`Ord3`** are variables that contain the sale amounts of the last three orders from a customer. Which of the following expressions can calculate the total of the two largest orders?

a. **`sum(max(of Ord1-Ord3),max(of Ord1-Ord3))`**

[✓ b.] **`sum(of Ord1-Ord3)-min(of Ord1-Ord3)`**

c. **`max(of Ord1-Ord3) + min(of Ord1-Ord3)`**

d. None of the above

**Adding the amount from all three orders and then subtracting the amount of the smallest order leaves the sum of the two largest orders.**