CHARACTER DATA – Acquisition, Manipulation, and Analysis

Andrew T. Kuligowski, HSN
Swati Agarwal, Optum
CHARACTER DATA – Acquisition, Manipulation, and Analysis

Andrew T. Kuligowski, HSN
Swati Agarwal, Optuminsight
Agenda

What is “Character Data”?  
How long is my Character Variable?

Manipulating Character Data

> Truncation Functions
> Concatenation Functions
> Length Functions
> Change Case Functions
> Substring Functions
> Misc. Functions
“Characters”?
What is “Character Data”?

“What character data consists of any combination of letters, symbols, and numeric characters.”

Quote from Microsoft documentation

**ASCII**
One byte per letter (upper and lower case), or number, or select special characters (the ones on a standard typewriter), plus a few non-printable: Carriage Return, Line Feed, Tab, Bell, etc.

**EBCDIC**
What is “Character Data”? 

“What character data consists of any combination of letters, symbols, and numeric characters.”

Quote from Microsoft documentation

**ASCII**

What about alphabetic characters used in other languages? Æ Ç Ñ

**EBCDIC**

What about special punctuation and symbols? ™ ¿

What about curly quotes? “ ”
What is “Character Data”?

“Character data consists of any combination of letters, symbols, and numeric characters.”

Quote from Microsoft documentation

Unicode

Many more characters can be defined and stored.

BUT … it takes more than one byte of storage to do so. (Nothing is free ...)
How Long is a Character Variable?

The default length of a character is 200 characters (unless, of course, overridden).
How Long is a Character Variable?

The default length of a character is 200 characters (unless, of course, overridden).

NOT ANYMORE

The MAXIMUM length of a character variable also used to be 200 bytes – now it's 32,767 bytes. *(THAT would be an undesirable default.)*
How Long is a Character Variable?

The default length of a character is now based on its first use. (Again, unless overridden.)

**Hardcoded Value**

The length of the FIRST value to which the variable is set (which may not be the longest value … well, for better or worse, it just became the longest value.)
How Long is a Character Variable?

The default length of a character is now based on its first use. (Again, unless overridden.)

<table>
<thead>
<tr>
<th>Value built from other character variable(s)</th>
<th>The TOTAL length of the variable(s) used to build the new variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you want to be safe, use a LENGTH statement – at the <em>top</em> of your DATA step.</td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>
How Long is a Character Variable?

data temp;
length w $ 2  y $ 40.;
retain w "";
x = "abcdefghijklmnopqrstuvwxyz";
y = "";
z = "1234567890";
output;
Y = "----+1----+2----+" ||
   "----+3----+4----+5";

a1 = substr(x,14);
a2 = substr(x,14,13);
a3 = x || y ;
output;
run;
How Long is a Character Variable?

You must define LENGTH before the first reference to the variable in your code. Otherwise ...

**WARNING:** Length of character variable Y has already been set. Use the LENGTH statement in the DATA STEP to declare the length of a character variable.

```
data temp;
  length w $ 2  y $ 40.;
  retain w "";
  x = "abcdefghijklmnopqrstuvwxyz";
  y = "";
  z = "1234567890";
  output;
  Y="----+----1----+----2----+----3----+----4----+----5"
```

```
a1 = substr(x,14);
a2 = substr(x,14,13);
a3 = x || y ;
output;
run;
```
How Long is a Character Variable?

 Longer one referenced first — Longer datasets when 2 or more

 Shorter one referenced first — <silence>

 WARNING: Multiple lengths were specified for the variable String
 by input data set(s). This may cause truncation of data.
Manipulation: Truncation Functions

- LEFT()
- RIGHT()
- TRIM()
- TRIMN()
- STRIP()
- COMPRESS()
- COMPBL()
- REPEAT()
- REVERSE()
Functions designed to aligned character variables:

- LEFT()
- RIGHT()

*No CENTERED() - you'll have to code that for yourselves.*
LEFT() vs. RIGHT()

```sas
data sample;
  set sample;
  left = '*' || left(string) || '*';
  right = '*' || right(string) || '*';
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>Left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe Smith</td>
<td>*Joe Smith</td>
<td>* Joe Smith*</td>
</tr>
<tr>
<td>2</td>
<td>Roma Brown</td>
<td>*Roma Brown</td>
<td>* Roma Brown*</td>
</tr>
<tr>
<td>3</td>
<td>Alice Wonde</td>
<td>*Alice Wonde</td>
<td>* Alice Wonde*</td>
</tr>
<tr>
<td>4</td>
<td>Li Wang</td>
<td>*Li Wang</td>
<td>* Li Wang*</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Note alignment in each value
TRIM() vs. TRIMN()

Functions designed to truncate character variables:

- TRIM()
- TRIMN()
- STRIP()
- COMPRESS()
data sample;
  input string $char14.;
datalines;
Joe Smith
  Roma Brown
  Alice Wonderland
  Li    Wang
;
contains trailing blanks
contains leading blanks
contains leading and trailing blanks
contains leading, trailing blanks and multiple blanks in between
(last line contains a blank string)
TRIM() vs. TRIMN()

data sample;
  input string $char14.;
datalines;
  Joe Smith
  Roma Brown
  Alice Wonderland
  Li Wang
;
  data sample;
  set sample;
  original = '*' || string || '*';
  trim = '*' || trim(string) || '*';
  trimn = '*' || trimn(string) || '*';
  run;
TRIM() vs. TRIMN()

<table>
<thead>
<tr>
<th>Original</th>
<th>Trim</th>
<th>Trimn</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Joe Smith</em></td>
<td><em>Joe Smith</em></td>
<td><em>Joe Smith</em></td>
</tr>
<tr>
<td><em>Roma Brown</em></td>
<td><em>Roma Brown</em></td>
<td><em>Roma Brown</em></td>
</tr>
<tr>
<td><em>Alice Wonde</em></td>
<td><em>Alice Wonde</em></td>
<td><em>Alice Wonde</em></td>
</tr>
<tr>
<td><em>Li Wang</em></td>
<td><em>Li Wang</em></td>
<td><em>Li Wang</em></td>
</tr>
</tbody>
</table>

Note difference in how blank value is handled.
Note lack of difference in how other values are handled.
STRIP() vs. TRIM() vs. TRIMN()

Functions designed to truncate character variables:

- **TRIM( )**
  - STRIP( ) vs. TRIM( LEFT( ) )
  - vs. TRIMN( LEFT( ) )
- **TRIMN( )**
  - Each will remove trailing blanks.
- **STRIP( )**
  - STRIP( ) will also remove leading blanks.
- **COMPRESS( )**
data sample;
set sample;
    strip = '*' || strip(string)       || '*';
    trim_left = '*' || trim(left(string))  || '*';
    trimn_left = '*' || trimn(left(string)) || '*';
run;

---

The SAS System
Output of STRIP, TRIM(LEFT) and TRIMN(LEFT)

<table>
<thead>
<tr>
<th>original</th>
<th>strip</th>
<th>trim_left</th>
<th>trimn_left</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Joe Smith</td>
<td><em>Joe Smith</em></td>
<td><em>Joe Smith</em></td>
<td><em>Joe Smith</em></td>
</tr>
<tr>
<td>* Roma Brown</td>
<td><em>Roma Brown</em></td>
<td><em>Roma Brown</em></td>
<td><em>Roma Brown</em></td>
</tr>
<tr>
<td>* Alice Wonde*</td>
<td><em>Alice Wonde</em></td>
<td><em>Alice Wonde</em></td>
<td><em>Alice Wonde</em></td>
</tr>
<tr>
<td>* Li Wang</td>
<td><em>Li Wang</em></td>
<td><em>Li Wang</em></td>
<td><em>Li Wang</em></td>
</tr>
</tbody>
</table>

No difference between STRIP and TRIMN( LEFT( ) ) (except efficiency)
COMPRESS() vs. COMPBL() 

- Functions designed to truncate character variables:
  - **TRIM( )**: COMPRESS removes a specified character(s) from a string. Default = “ ” if no characters specified.
  - **TRIMN( )**: COMPBL allows compression of multiple blanks into a single blank.
data sample;
  set sample;
  compress = '*' || compress(string) || '*';
  compbl = '*' || compbl(string) || ' '*;
run;

Note number and location of blanks in each value – including leading blanks.
data zipcode;
input zipcode $14.;
zipcode1 = compress(zipcode);
zipcode2 = compress(zipcode,' ()?');
zipcode3 = compress(zipcode,' - ()?');
zipcode4 = compress(zipcode,'ABCD','A');
zipcode5 = compress(zipcode,'23456','k');
datalines;
    22168-  12 34
    22168- (1234?)  1 – removes blanks
    MN55346 - mn44  2 – remove blanks, parens, question mark
                       3 – remove blanks, parens, question
                           marks, and dashes.
                       4 – REMOVE specified characters ABCD
                       5 – KEEP specified characters 23456
;
COMPRESS() vs. COMPBL() (1) removes blanks
2 – remove blanks, parens, question mark
3 – remove blanks, parens, question marks, and dashes.
4 – REMOVE specified characters ABCD
5 – KEEP specified characters 23456

The SAS System
Listing of Zipcodes

<table>
<thead>
<tr>
<th>zipcode</th>
<th>zipcode1</th>
<th>zipcode2</th>
<th>zipcode3</th>
<th>zipcode4</th>
<th>zipcode5</th>
</tr>
</thead>
<tbody>
<tr>
<td>22168- 12 34</td>
<td>22168-1234</td>
<td>22168-1234</td>
<td>221681234</td>
<td>22168- 12 34</td>
<td>226234</td>
</tr>
<tr>
<td>22168- (1234?)</td>
<td>22168-(1234?)</td>
<td>22168-1234</td>
<td>221681234</td>
<td>22168- (1234?)</td>
<td>226234</td>
</tr>
<tr>
<td>MN55346 - mn44</td>
<td>MN55346-mn44</td>
<td>MN55346-mn44</td>
<td>MN55346mn44</td>
<td>MN55346mn44</td>
<td>55346 - 44</td>
</tr>
</tbody>
</table>
Functions designed to aligned character variables:

- **REVERSE( )** Reverses SAS character expression, by default the variable length is determined by the length of the first argument.

- **REPEAT( )** Repeats the Character expression, by default the variable length is of 200 bytes.
data sample;
  set sample;
  backward = reverse(string);
  repeat   = repeat(string,1);
run;

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>backward</th>
<th>repeat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>eoJ</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Roma Brown</td>
<td>nworB amoR</td>
<td>Roma Brown Roma Brown</td>
</tr>
<tr>
<td>3</td>
<td>Alice</td>
<td>ecilA</td>
<td>Alice</td>
</tr>
</tbody>
</table>

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>backward</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>repeat</td>
<td>Char</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>string</td>
<td>Char</td>
<td>10</td>
</tr>
</tbody>
</table>
data sample;
  set sample;
  backward = reverse(string);
  repeat   = repeat(string,1);
run;

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>string</th>
<th>backward</th>
<th>repeat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>eoJ</td>
<td>Joe</td>
</tr>
<tr>
<td>2</td>
<td>Roma Brown</td>
<td>nworb amoR</td>
<td>Roma Brown Roma Brown</td>
</tr>
<tr>
<td>3</td>
<td>Alice</td>
<td>ecila</td>
<td>Alice</td>
</tr>
</tbody>
</table>

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th></th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>backward</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>repeat</td>
<td>Char</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>string</td>
<td>Char</td>
<td>10</td>
</tr>
</tbody>
</table>
Concatenation Functions

- CAT()
- CATT()
- CALL CATT Routine
- CATS()
- CALL CATS Routine
- CATX()
- CALL CATX Routine
Functions designed to concatenate character variables:

- **CAT()**
  - Concatenates character strings ...
  - ... without removing leading or trailing blanks.

- **CATT()**
  - ... and removes trailing blanks.

- **CATS()**
  - ... and removes leading and trailing blanks.

- **CATX()**
  - ... and removes leading and trailing blanks, and inserts separators between each string.
data sample;
  set sample;
  length cat catt cats $16 catx $20;
  text='Hello';
  cat = cat ('*', string, '*');  = ||
  catt = catt ('*', string, '*'); = TRIM || or TRIMN ||
  cats = cats ('*', string, '*'); = STRIP ||
  catx = catx ('!', text, string); = STRIP || separator
run;
Concatenation Functions

data sample;
    set sample;
    length cat catt cats $16 catx $20;
    text='Hello';
    cat =cat ('*',string,'*');
    catt=catt('*',string,'*');
    cats=cats('*',string,'*');
    catx=catx('!',text,string);
run;

The SAS System
Output of Concatenation Functions

cat | catt | cats | catx
---+-----+-----+-----
*Joe Smith * | *Joe Smith* | *Joe Smith* | Hello!Joe Smith
* Roma Brown * | *Roma Brown* | *Roma Brown* | Hello!Roma Brown
* Alice Wonde* | *Alice Wonde* | *Alice Wonde* | Hello!Alice Wonde
* Li Wang * | *Li Wang* | *Li Wang* | Hello!Li Wang
* ** | ** | ** | Hello

or TRIMN || separator
CAT functions are more efficient than the combination of TRIM( ) and || for concatenation.

NOTE: Default Length is 200.
CALL CATT ROUTINE: The CALL CATT routine returns the result in the first argument, result.

CALL CATT(result <, string-1, ...string-n>);

CALL CATS ROUTINE: The CALL CATS routine returns the result in the first argument, result.

CALL CATS(result <, string-1, ...string-n>);

CALL CATX ROUTINE: The CALL CATX routine returns the result in the second argument, result.

CALL CATX(separator, result<, string-1 , ...>string-n);
data _null_;  
    length catt $ 40 cats $ 40 catx $ 50;  
    x='Rio is t ';  
    y='he Olym ';  
    z='pic site for 2016.';  
    separator='&';  
    call catt(catt,x,y,z);  put catt;  
    call cats(cats,x,y,z);  put cats;  
    catx=catx(separator,catx,x,y,z);  put catx;  
run;
Manipulation : Length Functions

- LENGTH()
- LENGTHN()
- LENGTHC()
- LENGTHM()
Manipulation : Length Functions

Functions designed to report variable value length:
- **LENGTH( )** … excluding trailing blanks, and returns 1 for a blank string.
- **LENTHN( )** … excluding trailing blanks, and returns 0 for a blank string.
- **LENGTHC( )** … including trailing blanks.
- **LENGTHM( )** Returns the amount of memory (in bytes) that is allocated for a character string.
Manipulation : Length Functions

data how_long;
  one = 'SASGF2014 ';
  Two = ' ';
  length_one  = length(one);
  lengthn_one = lengthn(one);
  lengthc_one = lengthc(one);
  lengthm_one = lengthm(one);
  length_two  = length(two);
  lengthn_two = lengthn(two);
  lengthc_two = lengthc(two);
  lengthm_two = lengthm(two);
run;

Nonblank value with trailing blank. Missing value (Blank character).
Manipulation : Length Functions

\begin{verbatim}
data how_long;
one = 'SASGF2014 ';
Two = ' ';  Nonblank value with trailing blank.
length_one = length(one);
lengthn_one = lengthn(one);
lengthc_one = lengthc(one);
lengthm_one = lengthm(one);
length_two = length(two);
run;
\end{verbatim}

Missing value (Blank character).

<table>
<thead>
<tr>
<th>one</th>
<th>length_ one</th>
<th>lengthn_ one</th>
<th>lengthc_ one</th>
<th>lengthm_ one</th>
<th>length_ two</th>
<th>lengthn_ two</th>
<th>lengthc_ two</th>
<th>lengthm_ two</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASGF2014</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Manipulation : Change case Functions

- UPCASE()
- LOWCASE()
- PROPCASE()
Functions designed to change character values:

- **UPCASE( )** converts all uppercase letters to lowercase letters.
- **LOWCASE( )** converts all lowercase letters to uppercase letters.
- **PROPCASE( )** capitalizes the first letter of word in a string, leaving – or converting – the others to lower case.
Manipulation: Change Case Functions

```sas
data ds_1;
length Name $ 80.;
input Name $ &;
datalines;
rosy, mike and others
ROSY, MIKE AND OTHERS
ROSY, MIKE and OTHERS
;
```

```sas
data ds_2;
set ds_1;
/*convert it to other cases*/
upcase_var = upcase(Name);
propercase_var = propcase(Name);
lowcase_var =
    lowcase(upcase_var);
run;
```
Manipulation : Change case Functions

data ds_1;
length Name $ 80. ;
input Name $ & ;
datalines;
rosy, mike and others
ROSY, MIKE AND OTHERS
R
;

data ds_2;
set ds_1;
/*convert it to other cases*/
upcase_var = upcase(Name);
propercase_var = propcase(Name);
lowcase_var = propcase(upcase(Name));
run;

The SAS System
Listing of Names

<table>
<thead>
<tr>
<th>Name</th>
<th>upcase_var</th>
<th>propercase_var</th>
<th>lowcase_var</th>
</tr>
</thead>
<tbody>
<tr>
<td>rosy, mike and others</td>
<td>ROSY, MIKE AND OTHERS</td>
<td>Rosy, Mike And Others</td>
<td>rosy, mike and others</td>
</tr>
<tr>
<td>ROSY, MIKE AND OTHERS</td>
<td>ROSY, MIKE AND OTHERS</td>
<td>Rosy, Mike And Others</td>
<td>rosy, mike and others</td>
</tr>
<tr>
<td>ROSY, MIKE and OTHERS</td>
<td>ROSY, MIKE AND OTHERS</td>
<td>Rosy, Mike And Others</td>
<td>rosy, mike and others</td>
</tr>
</tbody>
</table>
Manipulation : Substring Functions

- SUBPAD Function
- SUBSTR (left of =) Function
- SUBSTR (right of =) Function
- SUBSTRN()
Manipulation : Substring Functions

SUBPAD( ) – will return a variable with the length specified, padding the results with spaces.

SUBSTR(left of = ) – Replaces Character value contents

SUBSTR(right of = ) – Extracts the substring

SUBSTRN( ) – will return a substring and allows with length of zero
Manipulation : Substring Functions

3529 DATA Substr_example;
3530 RETAIN Example "abcdefghijklmnopqrstuvwxyz";
3531 RETAIN Shortway "abcdefghijklmnopqrstuvwxyz";
3532 Longway = SUBSTR( Example, 1, 10 ) ||
  "123456" ||
  SUBSTR( Example, 17 ) ;
3535 SUBSTR( Shortway, 11, 6 ) = "123456" ;
3536 PUTLOG _ALL_;
3537 RUN;
Example =abcdefghijklmnopqrstuvwxyz
Shortway=abcdefghij123456qrstuvwxyz
Longway =abcdefghij123456qrstuvwxyz _ERROR_=0 _N_=1
Manipulation : Substring Functions

- **CHAR Function**
  
  \[
  \text{CHAR( string, position ) is equivalent to SUBSTR( string, position, 1 )}
  \]

- **FIRST Function**
  
  \[
  \text{FIRST( string ) is equivalent to SUBSTR( string, 1, 1 )}
  \]
Working with Character Data

MANY functions help find a character(s) within a character string:

INDEX( )
Search a character string for the presence of a specified string, return the 1st position of the latter within the former. (Search for “needle” in “haystack”.)

FIND( )
FIND searches for entire “needle”.

FINDC( )
FINDC searches for any character in “needle” that exists in “haystack”.

FINDW( )
FINDW searches for entire “needle” BUT it must be a “word” separated by a delimiter(s).
Working with Character Data

Many functions help find a character(s) within a character string:

- **INDEX( )**
- **FIND( )**
- **VERIFY( )**

**FIND** searches for the presence of a specified string, return the 1st position of the latter within the former.

(Find for “needle” in “haystack.”)

- **FINDC** searches for any character in “needle” that exists in “haystack.”
- **FINDW** searches for entire “needle” but it must be a “word” separated by a delimiter(s).

(“in my opinion”, “various modifications”)

**INDEX( )**

Characters string: Search a character string for any character, return the position of the character in the string.

(Search for “needle” in “haystack.”)

**FINDC**

Character string: Searches for any character in “needle” that exists in “haystack.”

(“in my opinion”)

**FINDW**

Character string: Searches for entire “needle” but it must be a “word” separated by a delimiter(s).

(“various modifications”)
Working with Character Data

MANY functions help find a character(s) within a character string:

INDEX( )
FIND( )
VERIFY( )

Search a character string for the presence of a character NOT in a specified string, return the 1st position of the latter within the former. (Search for “hay” in “haystack”, report back when you find something that is not “hay”.)
MANY functions help find a character(s) within a character string:

ANYALNUM()  ANYGRAPH()  ANYPUNCT()
ANYALPHA()   ANYLOWER()  ANYSPACE()
ANYCNTRL()   ANYNAME()   ANYUPPER()
ANYDIGIT()   ANYPRINT()  ANYXDIGIT()
ANYFIRST()

Search a character string for <something>, return the 1st position of the latter within the former.
Working with Character Data

MANY functions help find a character(s) within a character string:

NOTALNUM() NOTGRAPH() NOTPUNCT()
NOTALPHA() NOTLOWER() NOTSPACE()
NOTCNTRL() NOTNAME() NOTUPPER()
NOTDIGIT() NOTPRINT() NOTXDIGIT()
NOTFIRST()

Search a character string for <something>, return the 1st position of something that IS NOT the latter within the former.
COUNT() - counts substrings of characters in a character string.
COUNTC() - counts individual characters in a character string
IFC() - Returns a character value of an expression based on whether the expression is true, false, or missing
IFN() - Returns a numeric value of an expression based on whether the expression is true, false, or missing
data temp;
xyz='This is a thistle? Yes, this is a thistle.,'
howmanythis=count(xyz,'this');
howmanythis1=count(xyz,'this','i');
howmanyi = countc(xyz,'i');
run;

This is a thistle? Yes, this is a thistle.  3  4  6
data grade;
input name $ grade;
performance = ifc(grade>80, 'Pass', 'Needs Improvement');
if grade>80 then perf='Pass';
else perf = 'Needs Improvement';
datalines;
John 74
Kareem 89
Kati 100
Maria 92
;
run;

The SAS System 13:35

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>grade</th>
<th>performance</th>
<th>perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>74</td>
<td>Needs Improvement</td>
<td>Needs Improvement</td>
</tr>
<tr>
<td>2</td>
<td>Kareem</td>
<td>89</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>3</td>
<td>Kati</td>
<td>100</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>4</td>
<td>Maria</td>
<td>92</td>
<td>Pass</td>
<td>Pass</td>
</tr>
</tbody>
</table>
Manipulation : Misc. Functions

data sales;
input TotSales;
comm=ifn(TotSales > 10000, TotSales*.05, TotSales*.02);
if TotSales > 10000 then do comm_td = TotSales*.05;end;
else do;comm_td = TotSales*.02;end;
datalines;
25000
10000
500
10300
;
run;
Manipulation : Misc. Functions

MISSING() – If numeric and character expression contains missing value SAS returns ‘1’ else SAS returns ‘0’

TRANSLATE() – Replaces specific character

TRANWRD() - Replaces or removes all occurrences of a word in a character string
Manipulation : Misc. Functions

data name;
input @1 sal $5. @6 fname $6. @12 lname $20.;
datalines;
Miss  Joan  Smith
Ms   Ann
Miss Alice Cooper
;
run;
data show;
set name;
Sal_tranwrd = tranwrd(sal, "Miss", "Ms");
Sal_translate = translate(sal_tranwrd, "MS", "Ms");
if missing(lname) then comments = 'Last name is missing';
run;
data name;
input

<table>
<thead>
<tr>
<th>Obs</th>
<th>sal</th>
<th>fname</th>
<th>lname</th>
<th>Sal_tranwrd</th>
<th>Sal_translate</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Miss</td>
<td>Joan</td>
<td>Smith</td>
<td>Ms</td>
<td>MS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ms</td>
<td>Ann</td>
<td></td>
<td>Ms</td>
<td>MS</td>
<td>Last name is missing</td>
</tr>
<tr>
<td>3</td>
<td>Miss</td>
<td>Alice</td>
<td>Cooper</td>
<td>Ms</td>
<td>MS</td>
<td></td>
</tr>
</tbody>
</table>

run;

The SAS System 13:35 Monday,
data name;
input

The SAS System  13:35 Monday,

data show;
set name;
Sal_tranwrd = tranwrd(sal, "Miss", "Ms");
Sal_translate = translate(sal_tranwrd, "MS", "Ms");
if missing(lname) then comments = 'Last name is missing';
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>sal</th>
<th>fname</th>
<th>lname</th>
<th>Sal_tranwrd</th>
<th>Sal_translate</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Miss</td>
<td>Joan</td>
<td>Smith</td>
<td>Ms</td>
<td>MS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ms</td>
<td>Ann</td>
<td></td>
<td>Ms</td>
<td>MS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Miss</td>
<td>Alice</td>
<td>Cooper</td>
<td>Ms</td>
<td>MS</td>
<td>Last name is missing</td>
</tr>
</tbody>
</table>

Manipulation : Misc. Functions
Introduced in Version 9

More complex than “traditional” SAS functions.

More robust than “traditional” SAS functions.

Example: Search for a 3-character string where the first two characters are numbers and the 3rd is a 0.

```sas
IF _N_ = 1 THEN
   PATTERN_NEEDLE = PRXPARSE( "/\d\d0/" );

PRXMATCH( PATTERN_NEEDLE, STRING_HAYSTACK )
```
Working with Character Data
Ininformatics and Formats

$<nn>$. vs. $\text{CHAR}<nn>$. 

PROC FORMAT

PUT( ) function and
INPUT( ) function

ONLY QUICK OVERVIEW DUE TO TIME CONSTRAINTS.
Working with Character Data

Updates in recent releases

The K Functions are here!

KCOMPARE  KCOMPRESS  KCOUNT  KCVT
KINDEX   KINDEXC   KLEFT   KLENGTH
KLOWCASE  KREVERSE  KRIGHT  KSCAN
KSTRCAT  KSUBSTR  KSUBSTRB
KTRANSLATE
KUPCASE  KUPDATE  KUPDATEB  KVERIFY
Working with Character Data

Updates in recent releases

Version 9.3: \&SYSSIZEOFUNICODE.

Version 9.4: PUTC( ) and PUTN( ) now allow justification.

Version 9.4: SCAN( ) defaults resulting variable to length of 1st string processed.

Version 9.4: [NEW] TYPEOF( ) indicates whether an argument is character or numeric. (Only for WHERE clauses and Graphic Template Language [GTL].)
Working with Character Data

Updates in recent releases

Version 9.4: [NEW] `FCOPY( )` copies a record from one FILEREF to another.

Version 9.4: [NEW] `DOSUBL( )` trades values between SAS and the CALLING ENVIRONMENT's macro variables (not SAS macros!)

Version 9.4: [NEW] `SHA256( )` returns the result of a message digest of a specified string. (Cryptologists, take note!!)
Working with Character Data

This was an OVERVIEW of just 35 functions – designed to stimulate the curiosity, not to teach everything known about character data and how to work with it.
We hope it has done that – have fun as you continue to learn and to grow.

Contact:
swati_agarwal@optum.com
KuligowskiAndrew@gmail.com
Thank you for your interest in this topic!