**Chapter XIII – Transfeorming Data with SAS Functions**

1. General Form of SAS Functions

SAS functions can be used in DATA step programming statements. A SAS function can be specified anywhere that you would use a SAS expression

* **Arguments, Variable Lists, Arrays**

specify the function name followed by the function arguments, which are enclosed in parentheses

* Basic code:

**function-name*(****argument-1<,argument-n>****)*;**

*arguments* can be:

variables, mean(*x*,*y*,*z*)

constants, mean(*456*,*502*,*612*,*498*)

expressions, mean(37\*2,192/5,*mean*(*22*,*34*,*56*))

注意：Even if the function does not require arguments, the function name must still be followed by parentheses (for example, *function-name()*).

Eg: 1. contains multiple arguments

**MEAN**(x1,x2,x3)

2. variable list

**MEAN**(**OF** x1-x3)

3. array.

**MEAN**(**OF** newarray{\*})

注意：When assign array or list, keyword **OF** is needed

* Eg:

**DATA** hrd.newtemp;

**SET** hrd.temp;

Salary=payrate\*hours;

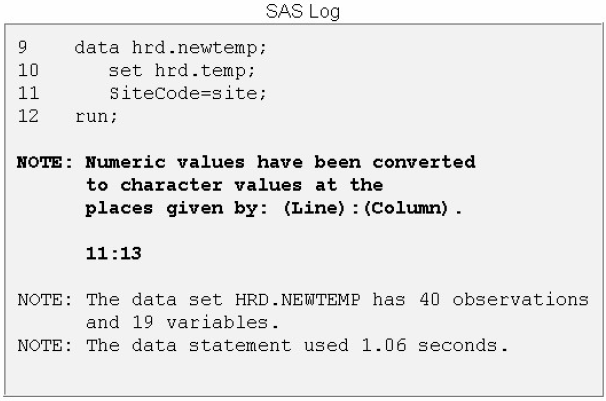
**RUN**;

* Target Variable: the variable to which the result of a function is assigned

1. Converting Data with Functions

When SAS detect mismatched variables, it will automatically convert it, but not always working well, so $ is needed. Include **INPUT** and **PUT** functions can avoid mismatches and circumvent automatic conversion

注意：这里的**INPUT**和**PUT**是function，有别于**INPUT**和**PUT** statement

* Automatically convert

Automatic numeric-to-character conversion also causes a message to be written to the SAS log indicating that the conversion has occurred.

* Nonautomatically convert
* **INPUT** function

Converts character data value to numeric values

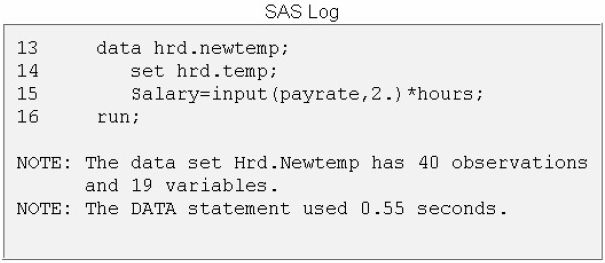
* Basic code:

**INPUT**(*source,informat*)

*source* indicates the character variable, constant, or expression to be converted to a numeric value

numeric *informat* must also be specified

* Eg:

当计算Salary=payrate\*hours时，PayRate（Originally character variable）可能会出现mismatch的现象，所以用INPUT function来convert character values of PayRate numeric values

**DATA** hrd.newtemp;

**SET** hrd.temp;

Salary=**INPUT**(payrate,2.)\*hours;

**RUN**;

* No conversion messages appear in the SAS log when you use the **INPUT**（so does **PUT**）function
* **PUT** function

Converts numeric values value to character data

在**INPUT**中使用informat，**PUT**中使用format

* Basic code:

**PUT**(*source*,*format*)

*source* indicates the numeric variable, constant, or expression to be converted to a character value

* The PUT function to create a variable that has not been previously identified, it creates a character variable whose length is equal to the format width.
* Concatenates numeric variable和character variable

The assignment statement that contains the concatenation operator (||) to indicate that Site should be concatenated with Dept,using a slash (/) as a separator. Note that the slash is enclosed in quotation marks. All character constants must be enclosed in quotation marks.

同时为了防止在numeric auto convert到character variable的过程中出现mismatch，**PUT** function会被使用：

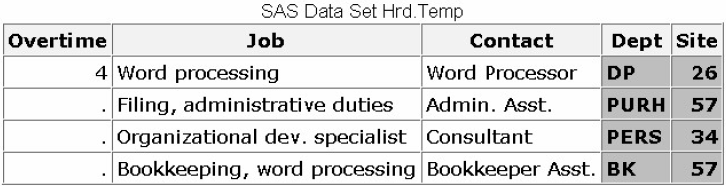
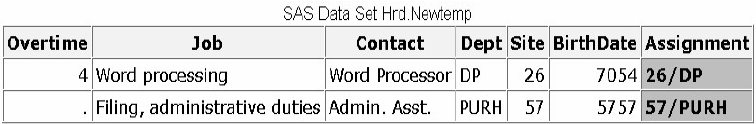
Eg:

**DATA** hrd.newtemp;

**SET** hrd.temp;

Assignment=site||'/'||dept;

**RUN**;

Before Concatenates After Concatenates

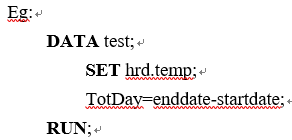
* Matching the data type

The format specified in the PUT function must match the data type of the source, so to do an explicit numeric-to-character data conversion, you specify a numeric source and a numeric format.

1. Manipulating SAS Date Values with Functions

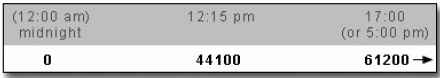
* SAS Date and Time Values

SAS stores date values as numbers so that you can easily sort the values or perform arithmetic computations

* SAS stores a date value as the number of days from January 1, 1960

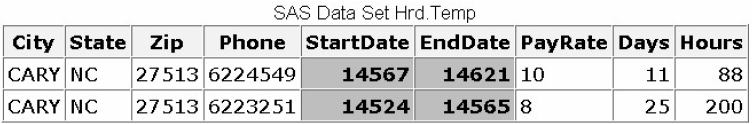


* SAS time value is stored as the number of seconds since midnight



* SAS datetime value is stored as the number of seconds between midnight on January 1, 1960, and a given date and time



* SAS can display date value in a variety of forms. The format affects *only* the display of the dates. A portion of the output created by this **PROC PRINT** step

Eg:

**PROC PRINT** data=hrd.temp;

**FORMAT** startdate enddate **DATE**9.;

**RUN**;

* SAS Date Functions
* Typical Use of SAS Date Functions

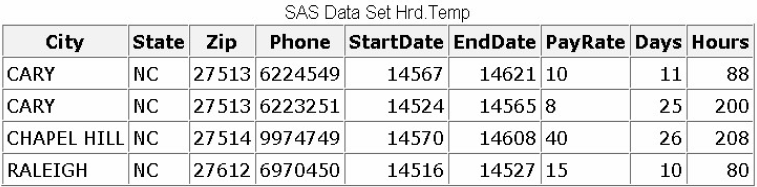
|  |  |  |
| --- | --- | --- |
| Function | Typical Use | Result |
| **MDY** | date = **MDY**(mon, day, yr); | SAS date |
| **TODAYDATE** | now = today(); | today's date as a SAS date |
|  | now = date(); |  |
| **TIME** | curtime=time(); | current time as a SAS time |

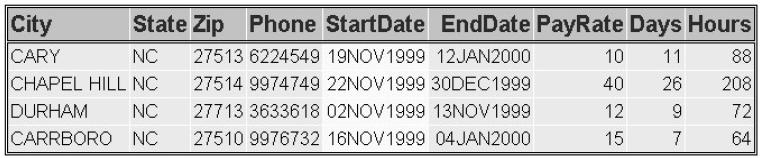
* Selected Functions to Use with SAS Date Values

|  |  |  |
| --- | --- | --- |
| Function | Typical Use | Result |
| **DAY** | day = **DAY**(date); | Day of month(1-31) |
| **QTR** | quarter = **QTR**(date); | Quarter(1-4) |
| **WEEKDAY** | wkday = **WEEKDAY**(date); | Day of week(1-7) |
| **MONTH** | month = **MONTH**(date); | Month(1-12) |
| **YEAR** | yr = **YEAR**(date); | Year(4 digits) |
| **INTCK** | x = **INTCK**(‘day’, d1, d2); | Days from D1 to D2 |
|  | x = **INTCK**(‘week’, d1, d2); | Week from D1 to D2 |
|  | x = **INTCK**(‘month’, d1, d2); | Month from D1 to D2 |
|  | x = **INTCK**(‘qtr’, d1, d2); | Quarter from D1 to D2 |
|  | x = **INTCK**(‘year’, d1, d2); | Year from D1 to D2 |
| **INTNX** | x = **INTNX**(‘interval’, start-from, increment); | Date, time, or datetime value |
| **DATDIF** | x = **DATDIF**(date1, date2, ‘ACT/ACT’); | Days between date1 and date2 |
| **YRDIF** | x = **YRDIF** (date1, date2, ‘ACT/ACT’); | Years between date1 and date2 |

* Eg: Finding the Year and Month

Suppose you need to create a subset of the data set Hrd.Temp that contains information about all temporary employees who were hired in November 1999. Hrd.Temp contains the beginning and ending dates for staff employment, but there are no month or year variables in the data set.



* To determine the year and month in which employees were hired, you can apply the **YEAR** and **MONTH** function to the variable that contains the employee start date, StartDate
* To create the new data set, you include these functions in a subsetting **IF** statement within a **DATA** step. The subsetting **IF** statement specifies that the new data set includes only observations in which the **YEAR** function extracts a value of **1999** and the **MONTH** function extracts a value of **11** (for November).
* Finally, you add a **PROC** **PRINT** step to the program so that you can view the new data set. Notice that the **PROC** **PRINT** step includes a **FORMAT** statement to display the variables StartDate and EndDate with the **DATE**9. format.

**DATE** hrd.nov99;

**SET** hrd.temp;

**IF** year(startdate)=1999 and

month(startdate)=11;

**RUN**;

**PROC** **PRINT** data=hrd.tempnov;

**FORMAT** startdate enddate **DATE**9.;

**RUN**;

* **WEEKDAY** function

Enable you to extract the day of the week from a SAS data value

* Basic code:

**WEEKDAY(***date***)**

*date* is a SAS date value that is specified either as a variable or as a SAS date constant

* Values for the WEEKDAY Function

|  |  |
| --- | --- |
| Value | Day of the week |
| 1 | Sunday |
| 2 | Monday |
| 3 | Tuesday |
| 4 | Wednesday |
| 5 | Thursday |
| 6 | Friday |
| 7 | Saturday |

* Eg:

To create a data set that contains only weekend broadcasts, you use the WEEKDAY function in a subsetting IF statement

**DATA** radio.schwkend;

**SET** radio.sch;

**IF** **WEEKDAY**(airdate)**IN**(1,7);

**RUN**;

注意：the statement if WEEKDAY(airdate) IN (1,7); is the same as if WEEKDAY(airdate)=7 or WEEKDAY(airdate)=1;

* **MDY** function

The **MDY** function creates a SAS date value from numeric values that represent the month, day, and year.

如果在**MDY** function中输入错误的月日年数据，result会将错误的日期显示为missing。

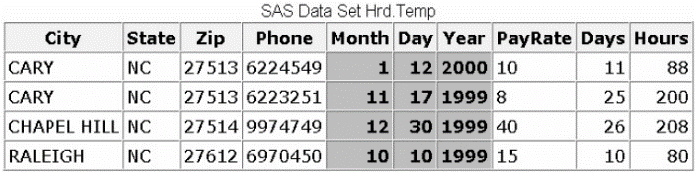
* Basic code:

*month* can be a variable that represents the month, or a number from 1-12

*day* can be a variable that represents the day, or a number from 1-31

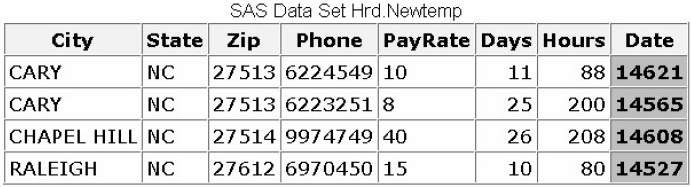
*year* can be a variable that represents the year, or a number that has 2 or 4 digits.

**MDY(***month,day,year***)**

* Eg：

**DATA** hrd.newtemp(DROP=month day year);

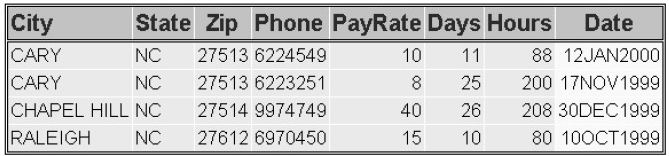
**SET** hrd.temp;

**Date**=mdy(month,day,year);

**RUN**;

**PROC PRINT** data=hrd.newtemp;

**FORMAT** date **DATE**9.;

**RUN**;

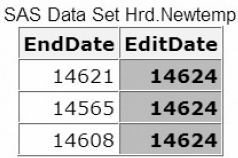
注意：使用**MDY** function时，如果使用2 digits year value时，SAS会按照100-year span来计算(**YEARCUTOFF**= option)。系统默认**YEARCUTPFF**= 1920。建议使用4 digits year。

* DATE() and TODAY() function

Return the current date from the system clock as a SAS date value

Eg:

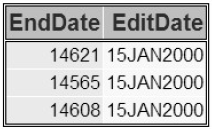
The SAS date values shown below were created by submitting this program on January 15, 2000.

**DATA** hrd.newtemp;

**SET** hrd.temp;

EditDate=**DATE**();

**RUN**;

**PROC PRINT** data=hrd.newtemp;

**FORMAT** editdate **DATE**9.;

**RUN**;

* **INTCK** function

Returns the number of time intervals that occur in a given time span, can be use to identify periodic events such as due dates and anniversaries

* Basic code:

**INTCK(***'interval',from,to***)**

*'interval'* specifies a character constant or variable. The value can be one of the following:

DAY, WEEKDAY, WEEK, TENDAY, SEMIMONTH, MONTH, QTR, SEMIYEAR, YEAR, DATETIME, TIME

*From* specifies a SAS date, time, or datetime value that identifies the beginning of the time span.

*to* specifies a SAS date, time, or datetime value that identifies the end of the time span.

注意：type of interval必须和from的type相同

* Counts intervals from fixed interval beginnings, not in multiples of an interval unit *from* the from value. Partial intervals are not counted.

Eg:

WEEK intervals are counted by Sundays rather than seven-day multiples from the *from* argument. MONTH intervals are counted by day 1 of each month, and YEAR intervals are counted from 01JAN, not in 365-day multiples

|  |  |
| --- | --- |
| SAS Statement | SAS Statement |
| Weeks=intck('week','31dec2000'd,'01jan2001'd); | 0 |
| Months=intck('month','31dec2000'd,'01jan2001'd); | 1 |
| Years=intck('year','31dec2000'd,'01jan2001'd); | 1 |

Because December 31, 2000, is a Sunday, no WEEK interval is crossed between that day and January 1, 2001. However, both MONTH and YEAR intervals are crossed.

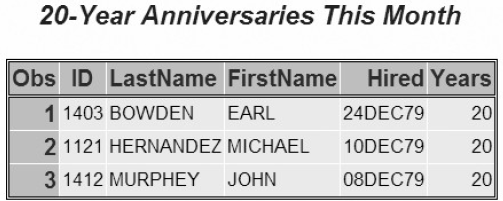
* Eg:

The following program identifies mechanics whose 20th year of employment occurs in the current month.

Output is created when the program is run in December of 1999.

**DATA** work.anniv20;

**SET** flights.mechanics(KEEP=id lastname firstname hired);

Years=**INTCK**('year',hired,today());

**IF** years=20 and month(hired)=month(TODAY());

**RUN**;

**PROC** **PRINT** data=work.anniv20;

**TILTLE** '20-Year Anniversaries This Month';

**RUN**;

* **INTNX** function

It’s similar to **INTNK**, which applies multiples of a given interval to a date, time, or datetime value and returns the resulting value can use the **INTNX** function to identify past or future days, weeks, months, and so on

* Basic code:

**INTNX(***'interval',start-from,increment*<,'*alignment*'>**)**

*'interval'* specifies a character constant or variable (Same as **INTNK**)

*start-from* specifies a starting SAS date, time, or datetime value

*increment* specifies a negative or positive integer that represents time intervals toward the past or future

'*alignment*'(optional) forces the alignment of the returned date to the beginning, middle, or end of the interval.

* BEGINNING Alias: B
* MIDDLE Alias:M
* END Alias:E
* SAME Alias: SAMEDAY or S
* Alignment Values for the INTNX Function

|  |  |
| --- | --- |
| SAS Statement | Date Value |
| MonthX=intnx('month','01jan1995'd,5,'*b*'); | 12935 (June 1, 1995) |
| MonthX=intnx('month','01jan1995'd,5,'*m*'); | 12949 (June 15, 1995) |
| MonthX=intnx('month','01jan1995'd,5,'*e*'); | 12964 (June 30, 1995) |

These **INTNX** statements count five months from January, but the returned value depends on whether alignment specifies the beginning, middle, or end day of the resulting month. If alignment is not specified, the beginning day is returned by default.

* **DATDIF** and **YRDIF** Functions

Calculate the difference in days and years between two SAS dates

* Basic code:

**DATDIF(***start\_date,end\_date,basis)***)**

**YRDIF(***start\_date,end\_date,basis)***)**

*start\_date* specifies the starting date as a SAS date value

*end\_date* specifies the ending date as a SAS date value

*basis* specifies a character constant or variable that describes how SAS calculates the date difference.

* Character Strings in the DATDIF Function

|  |  |  |  |
| --- | --- | --- | --- |
| **Character String** | **Meaning** | **Valid in DATDIF** | **Valid in YRDIF** |
| ‘30/360’ | specifies a 30 day month and a 360 day year | YES | YES |
| ‘ACT/ACT’ | uses the actual number of days or years between dates | YES | YES |
| ‘ACT/360’ | uses the actual number of days between dates in calculating  the number of years (calculated by the number of days divided  by 360) | NO | YES |
| ‘ACT/365’ | uses the actual number of days between dates in calculating  the number of years (calculated by the number of days divided  by 365) | NO | YES |

* Eg:

|  |  |
| --- | --- |
| **YRDIF Statement** | **Returned Value** |
| **YRDIF** ('16oct1998'd,'16feb2003'd,'30/360') | 4.333333333 |
| **YRDIF** ('16oct1998'd,'16feb2003'd,'ACT/ACT') | 4.3369863014 |
| **YRDIF** ('16oct1998'd,'16feb2003'd,'ACT/360') | 4.4 |
| **YRDIF** ('16oct1998'd,'16feb2003'd,'ACT/365') | 4.3397260274 |

1. Modifying Character Values with Functions

To replace the contents of a character value, trim trailing blanks from a character value, search a character value and extract a portion of the value, convert a character value to uppercase or lowercase.

* Selected Character Functions （50题，Q21）

|  |  |
| --- | --- |
| Function | Purpose |
| **SCAN** | returns a specified word from a character value |
| **SUBSTR** | extracts a substring or replaces character values |
| **TRIM** | trims trailing blanks from character values |
| **CATX** | concatenates character strings, removes leading and trailing blanks, and inserts  separators |
| **INDEX** | searches a character expression for a string of characters, and returns the position of the string's first character for the first occurrence of the string |
| **FIND** | searches for a specific substring of characters within a character string |
| **UPPERCASE** | converts all letters in a value to uppercase |
| **LOWCASE** | converts all letters in a value to lowercase |
| **PROPCASE** | converts all letters in a value to proper case |
| **TRANWRD** | replaces or removes all occurrences of a pattern of characters within a character string |

* **SCAN** function

Enables you to separate a character value into words, to return a specified word, and to create new variables. (拆分character values)

* Delimiters
* The SCAN function uses delimiters, which are characters that are specified as word separators, to separate a character string into words (can have multiple delimiters in same **SCAN** function)
* Default delimiters

blank . < ( + | & ! $ \* ) ; ^ - / , %

* Basic code:

**SCAN(***argument,n*<,delimiters>**)**

where

* *argument* specifies the character variable or expression to scan.
* *n* specifies which word to return.
* delimiters are special characters that must be enclosed in single quotation marks (' '). If you do not specify *delimiters*, default delimiters are used.
* Specifying Variable Length

**SCAN** function assigns a length of 200 to each target variable, usually the default length is way longer than necessary for variables

* Eg:

**DATA** Hrd.Newtemp(**DROP**=name);

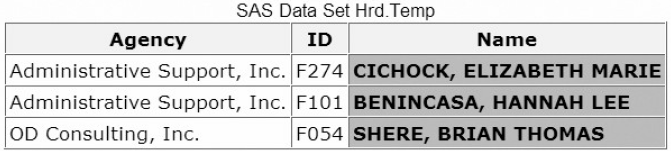
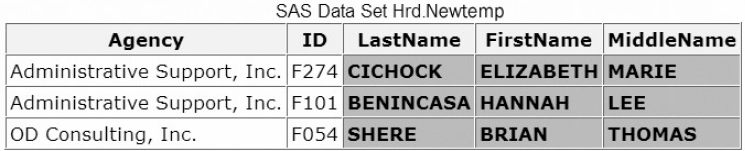
**SET** Hrd.Temp;

**LENGTH** LastName FirstName MiddleName $ 10;

LastName=**SCAN**(name,1);

FirstName=**SCAN**(name,2);

MiddleName=**SCAN**(name,3);

**RUN**;

* Different between **SCAN** and **SUBSTR**
* **SCAN** extracts words within a value that is marked by delimiters.
* **SUBSTR** extracts a portion of a value by starting at a specified location.
* **SUBSTR** function

To extract a portion of a character value (any number), and to replace the contents of a character value.

* Basic code:

**SUBSTR (***argument,position* <,n>**)**

*argument* specifies the character variable or expression to scan.

*position* is the character position to start from.

n specifies the number of characters to extract. If *n* is omitted, all remaining characters are included in the substring.

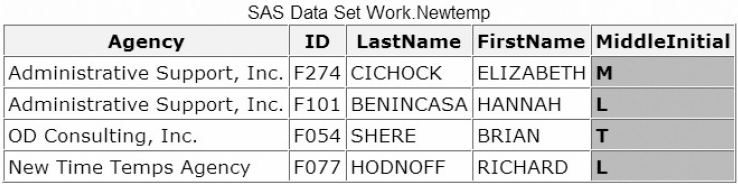
* Eg:

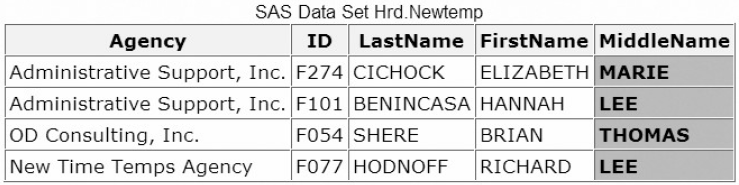
This function extracts a character string from the value of MiddleName. The string to be extracted begins in position 1 and contains one character. Then, you place this function in an assignment statement in your DATA step.

**DATA** Work.Newtemp(DROP= MiddleName);

**SET** Hrd.Newtemp;

**LENGTH** MiddleInitial $ 1;

MiddleInitial= **SUBSTR** (MiddleName,1,1);

**RUN**;

* Replacing Text Using **SUBSTR**
* Comparison:

When the function is on the ***right******side***of an assignment statement, the function returns the requested string.

MiddleInitial= **SUBSTR** (middlename,1,1);

But if you place the **SUBSTR** function on the ***left******side***of an assignment statement, the function is used to modify variable values.

**SUBSTR** (region,1,3)='NNW';

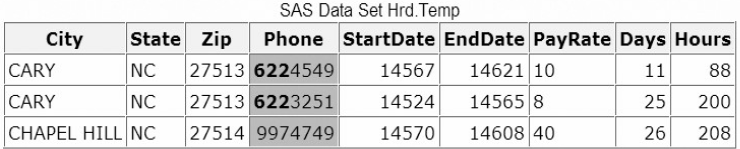
* Eg:

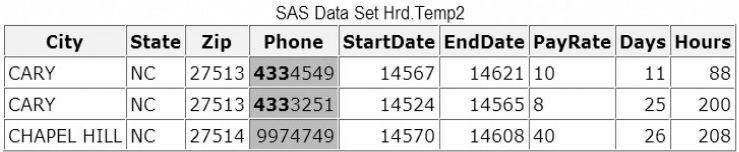
To replace the 622 exchange in the variable Phone, but you needs an **IF-THEN** statement to verify the value of the variable Exchange. If the exchange is 622, the assignment statement executes to replace the value of Phone.

**DATA** Hrd.Temp2(**DROP**=exchange);

**SET** Hrd.Temp;

Exchange=substr(phone,1,3);

**IF** exchange='622' **THEN** substr(phone,1,3)='433';

**RUN**;

* **TRIM** Function

Enables you to remove trailing blanks from character values (合并character values)

The concatenation operator (||) enables you to concatenate character values, and commas and blanks are needed to separate values.

The length of new variable is the sum of the length of each variable and constant that is used to create the new variable.

* Basic code:

**TRIM(***argument***)**

*argument* can be any character expression, such as

character variable: trim(address)

either character function: trim(left(id)).

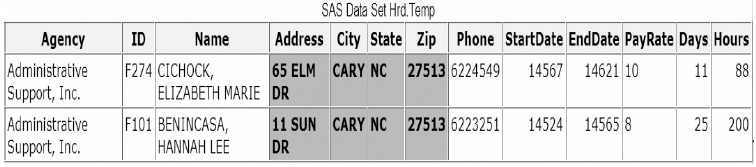
* Eg:

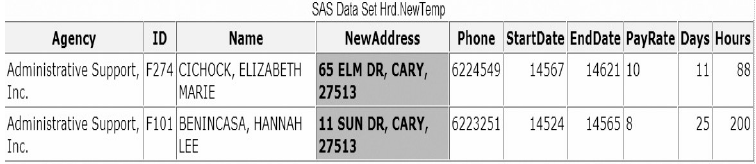
**DATA** Hrd.Newtemp(**DROP**=address city state zip);

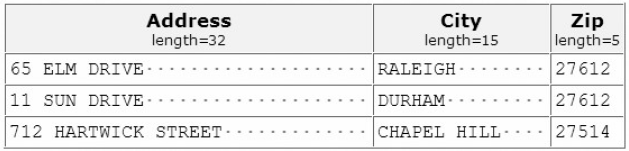
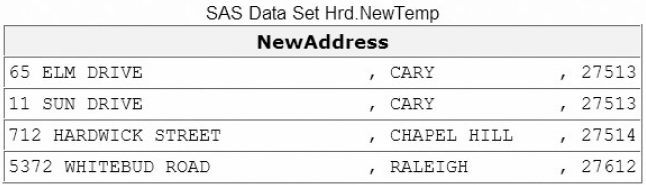
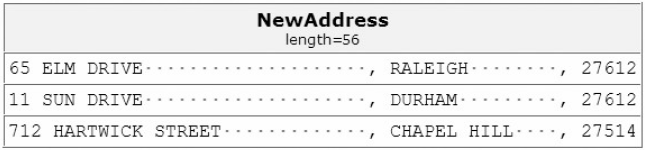
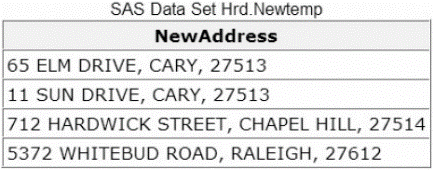
**SET** Hrd.Temp;

NewAddress=**TRIM**(address)||', '|| **TRIM** (city)||', '||zip;

**RUN**;

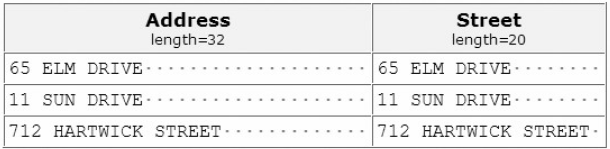




* 注意：因为original variable含有台铃blanks，所以当合并旧的variables时会出现。**TRIM** function enables you to remove trailing blanks from character values.
* 注意：The TRIM function does not affect how a variable is stored. The trimmed values are padded with trailing blanks again if the values are shorter than the length of the new variable

Eg:

When the trimmed value is assigned to Street, trailing blanks are added to the value to match the length of 20.

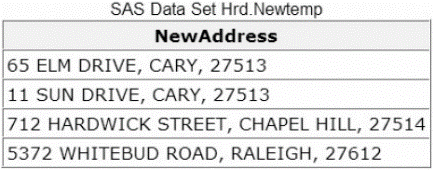
**DATA** temp;

**SET** hrd.temp;

**LENGTH** Street $ 20;

Street=**TRIM**(address);

**RUN**;

* **CATX** Function
* Enables you to concatenate character strings, remove leading and trailing blanks, and insert separators, which returns a value to a variable, or returns a value to a temporary buffer.
* Usually the result of **CATX** is equivalent to those that are produced by a combination of the concatenation operator and the **TRIM** and **LEFT** functions.
* The default length of **CATX** Function is 200 bytes, add a **LENGTH** statement to your **DATA** step, and specify an appropriate length for your variable. But be sure to place the **LENGTH** statement before the assignment statements that contain the **CATX** function
* Eg:

**DATA** Hrd.Newtemp(**DROP**=address city state zip);

**SET** Hrd.Temp;

NewAddress=**CATX**(', ',address,city,zip);

**RUN**;

* **INDEX** Function

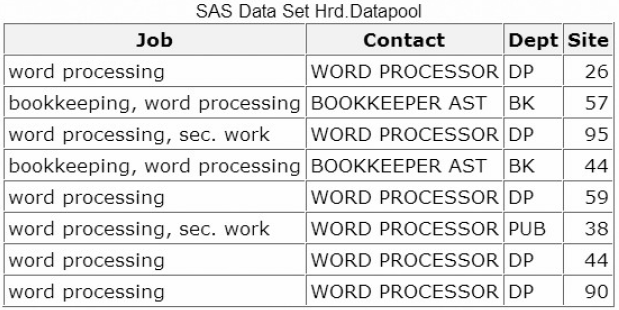
Enables you to search a character value for a specified string. The **INDEX** function searches values from left to right, looking for the first occurrence of the string. It returns the position of the string's first character; if the string is not found, it returns a value of **0**

* Basic code:

**INDEX(***source,excerpt***)**

*source* specifies the character variable or expression to search

*excerpt* specifies a character string that is enclosed in quotation marks ('')

* 如果只想让SAS显示match的variables，我们可以加一个**IF** statement来筛选出所有搜寻结果大于0的项
* Eg:

**DATA** Hrd.Datapool;

**SET** Hrd.Temp;

**IF** **INDEX**(job,'word processing') > 0;

**RUN**;

* 注意：在搜寻中，character string必须和所搜寻的词组exactly match，其中包含大小写。如果上述例子中search的词组为'WORD PROCESSING'，显示结果将是no match或者0
* To ensure all occurrences of characters string are found, use **UPCASE** and **LOWCASE** function with the **INDEX** function. 这两个公式吧所有的characters string统一转换为同一个case，以便运用**INDEX** function

Eg:

**INDEX** (**UPCASE** (job),'WORD PROCESSING')

**INDEX** (**LOWCASE** (job),'word processing')

* **FIND** function (similar to **INDEX** function)

Enables you to search for a specific substring of characters within a character string that you specify. It searches a string for the first occurrence of the substring, and returns the position of that substring

* Basic code:

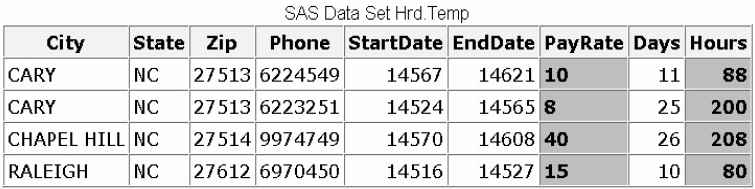
**FIND**(*string*,*substring*<,*modifiers*><,*startpos*> )

*string* specifies a character constant, variable, or expression that will be searched for substrings

*substring* is a character constant, variable, or expression that specifies the substring of characters to search for in string

*modifiers* is a character constant, variable, or expression that specifies one or more modifiers

*startpos* is an integer that specifies the position at which the search should start and the direction of the search. The default value for startpos is 1.

* 注意：
* modifier让function可以在搜索中忽略大小写，如果没有加入modifier，**FUND** function会搜做和所搜寻substring相同case的character
* 如果modifier是一个常数，需将modifier置于quotation marks，可以在一个quotation marks中置入多个constants modifier. Modifier values are not case sensitive
* 如果startpos没有specify，那么搜索将会从string的开头开始，并且搜索整个string；但是如果specify了startpos后，startpos将是search的起点。如果startpos为positive，那么**FIND** searches from startpos to the right; 如果为*negative*, **FIND** searches from startpos to the left.
*  Eg:

**DATA** Hrd.Datapool;

**SET** Hrd.Temp;

**IF** **FIND**(job,'word processing') > 0;

**RUN**;

* **UPCASE & LOWCASE** Function

Converts all letters in a character expression to uppercase/lowercase

* Basic code:

**UPCASE**(*argument*)

*argument* can be any SAS character expression, such as a character variable or constant.

**LOWCASE(***argument***)**

*argument* can be any SAS character expression, such as a character variable or constant

* Eg:

**DATA** Hrd.NewTemp;

**SET** Hrd.Temp;

Job=**UPCASE**(job);

**RUN**;

* **PROPCASE** function

Converts all words in an argument to proper case (so that the first letter in each word is capitalized).

* Basic code:

**PROPCASE(***argument*<*,delimiter(s)*>**)**

*argument* can be any SAS expression, such as a character variable or constant

*delimiter (s)* specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

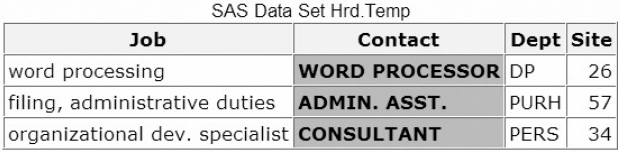
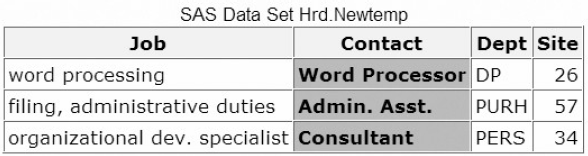
* 注意：**PROPCASE** uses the default delimiters unless you use the delimiter(s) argument
* Eg:

**DATA** Hrd.NewTemp;

**SET** Hrd.Temp;

Contact=**PROCASE**(contact);

**RUN**;



* **TRANWRD** Function
* Replaces or removes all occurrences of a pattern of characters within a character string, and the translated characters can be located anywhere in the string
* The default length is 200 bytes, if you want to save storage space, a **LENGTH** statement can be adding before the **TRANWRD** function in the **DATA** step
* Basic code:

**TRANWRD(***source,target,replacement***)**

*source* specifies the source string that you want to translate

*target* specifies the string that SAS searches for in *source*

*replacement* specifies the string that replaces *target*.

注意：*target*和*replacement*可以是variables或者character strings。如果是character strings，必须enclose the string in quotation marks

* Eg:

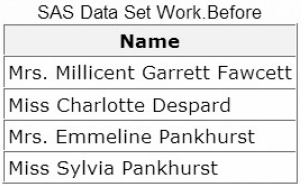
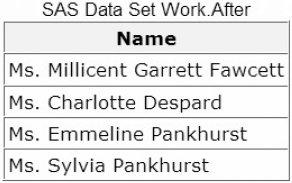
**DATA** Work.After;

**SET** Work.Before;

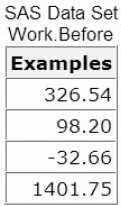
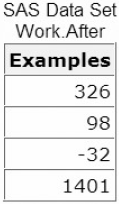
name= **TRANWRD** (name,'Miss','Ms.');

name= **TRANWRD** (name,'Mrs.','Ms.');

**RUN**;

1. Modifying Numeric Values with Functions

* **INT** Function

To return the integer portion of a numeric value, decimal will be discarded

* Basic code:

**INT**(*argument*)

*argument* is a numeric variable, constant, or expression.

* Eg:

**DATA** Work.After;

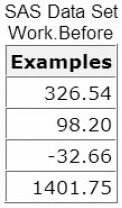
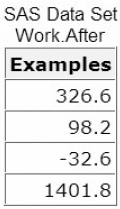
**SET** Work.Before;

Examples=**INT**(Examples);

**RUN**;

* **ROUND** Function

To round values to the nearest specified unit

* Basic code:

**ROUND(***argument,round-off-unit***)**

*argument* is a numeric variable, constant, or expression.

*round-off-unit* is numeric and nonnegative.

注意：If a round-off unit is not provided, a default value of 1 is

used, and the argument is rounded to the nearest integer.

* Eg:

**DATA** Work.After;

**SET** Work.Before;

Examples=**ROUNF**(Examples,.2);

**RUN**;

**练习**

1. Suppose you need to create the variable FullName by concatenating the values of FirstName, which contains first names, and LastName, which contains last names. What's the best way to remove extra blanks between first names and last names?

