Calculating Age in One Line of Code
Wei Wang, Highmark Blue Cross Blue Shield, Pittsburgh, PA

ABSTRACT
There are many ways of calculating a person’s age based on a predefined
date and the person’s birthday. Some methods fail to calculate a correct
age, particularly when the birthday falls on a leap year. Oftentimes, a
precise calculation is necessary. This paper presents a single line
solution to accurately calculate age that takes into account leap year and
non-leap year birthdays.

INTRODUCTION
Exact computation of ages must take into account all rules governing the
length of months and the length of years. William Kreuter presented an
efficient one line code. In his article, Accurately Calculating Age with Only
One Line of Code, Kreuter discussed the two instances his code might fail to yield the expected result. One of them is when the birthday is
February 29, and during non-leap years the person celebrates the
birthday on February 28. This paper presents a modified solution to
Kreuter’s code that takes into consideration the leap year problem.

THE “MAGIC” FUNCTIONS: INTCK AND INTNX
The function

\[
\text{INTCK}(\text{interval}, \text{from}, \text{to})
\]
returns an integer representing the number of intervals that occur in a
given time span.

For the following example:

\[
\text{mon} = \text{intck}('\text{month}', '10\text{oct}88'd, '01\text{mar}89'd);
\]
returns a value of 5.

The function

\[
\text{INTNX}(\text{interval}, \text{from}, \text{number})
\]
generates a SAS date, time, or datetime value that is a given number of
intervals from a starting value, indicated by from.

For the following example:

\[
X = \text{intnx}('\text{month}', '05\text{jan}90'd, 1);
\]
Returns February 1, 1990.

HOW INTCK CALCULATES AGE
Let’s look at the function

\[
\text{intck}(\text{’month’}, \text{birthday}, \text{someday})
\]
It returns how many times the first day of a month is passed between
birthday and someday.

In order to get the number of times the same day of the month is passed,
we need to modify the above code to:

\[
\text{intck}(\text{’month’}, \text{birthday}, \text{someday}) -
(\text{day(someday) } < \text{day(birthday)})
\]

This now gives exactly the number of months for any given
birthday and someday.

Converting months to years, we get:

\[
\text{age} = \text{floor} \left( \frac{\text{intck}('\text{month}', \text{birthday}, \text{someday}) - (\text{day(someday) } < \text{day(birthday)} \)}{12} \right);
\]

The solution described here would treat a February 29 birthday
during non-leap years as March 1.

TAKING CARE OF LEAP YEAR
If someone’s birthday is on Feb 29 of a leap year and we are
calculating the age on a non-leap year,

\[
\text{day(someday) } < \text{day(birthday)}
\]
would return 1 if someday is Feb 28, and 0 if someday is
March 1, which is not what we want.

Let’s look at the function

\[
\text{intnx('month', someday, 1) } - 1
\]
it will return the last day of the month! In particular, if someday
is in February, it will return Feb 28, or Feb 29 depending on
whether it is leap year.

Add one more step

\[
\text{min( day(birthday), day(intnx (’month’, someday, 1) } -1 \})
\]
For Feb 29 birthdays, when someday is in February, this will
return 28, during non-leap years and 29 during the leap years.

Thus, we convert a Feb 29 birthday into Feb 28 during non-
leap years.

Combining the above code with the age calculation code, we get:

\[
\text{age} = \text{floor} \left( \frac{\text{intck}('\text{month}', \text{birthday}, \text{someday}) - \text{day(someday) } < \min (\text{day(birthday), day(intnx (’month’, someday, 1) } -1 \})}{12} \right);
\]

This can be conveniently set up as a macro:

\%
macro age (birthday, someday);
  \text{floor} \left( \frac{\text{intck}('\text{month}', \text{&birthday}, \text{&someday}) - \text{day(\&someday) } < \min (\text{day(\&birthday), day(intnx (’month’, \text{&someday, 1) } -1 \})}{12} \right)
%end

The following is an example of how the macro can be used in a
SAS DATA step:

\%
\text{age} = %age (today(), birth);
%end

Because it is in one line of code, it also can be used in proc
sql’s select statement:
Proc sql;
Create table ages as
Select %age(today(), birth) as age
From birthset;
Quit;

TESTING THE CODE
The code can be tested as:

```data _null_;
age1 = %age ('29FEB1992'D, '28FEB1998'D);
age2 = %age ('29FEB1992'D, '01MAR1998'D);
put age1= age2=;
run;
```

```data _null_;
age1 = %age ('28FEB1993'D, '28FEB1996'D);
age2 = %age ('01MAR1993'D, '01MAR1996'D);
put age1= age2=;
run;
```

```data _null_;
age1 = %age ('31JAN1990'D, '31JAN2000'D);
age2 = %age ('30JAN1990'D, '30JAN2000'D);
put age1= age2=;
run;
```

The results will be placed on the log:

```
age1=6
age2=6
```
```
age1=3
age2=3
```
```
age1=10
age2=10
```

respectively.

CONCLUSION
The code presented here provide an easy way to calculate a person’s age. It can be used in data step and proc sql's select statement.

REFERENCES
   http://www.sas.com/service/techtips/quicktips/calcage.html

ACKNOWLEDGMENTS
Thanks to Rick Pro and John Newell for their support and encouragement. Special thanks to John Newell and David Conti for their reviews and suggestions.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Wei Wang
Highmark Blue Cross Blue Shield
120 Fifth Avenue `Ste P7205
Pittsburgh, Pa 15222-3099
Work: (412)544-0748
Fax: (412)544-0700
Email: WEI.WANG@HIGHMARK.COM