

Macros

Statistics 426: SAS Programming

Module 11

2021

Functions (commands)

Sometimes there are calculation tasks that must be performed again and again; SAS contains many useful calculation tasks programmed into procedures (PROCs) and other (stand-alone) functions that can be used to create macros, user-defined procedures

A function is usually designed to take a quantity or a vector, *called the argument of the function*, and calculate something with it; it returns the value of the calculation as output for further use

The general form for a function is:

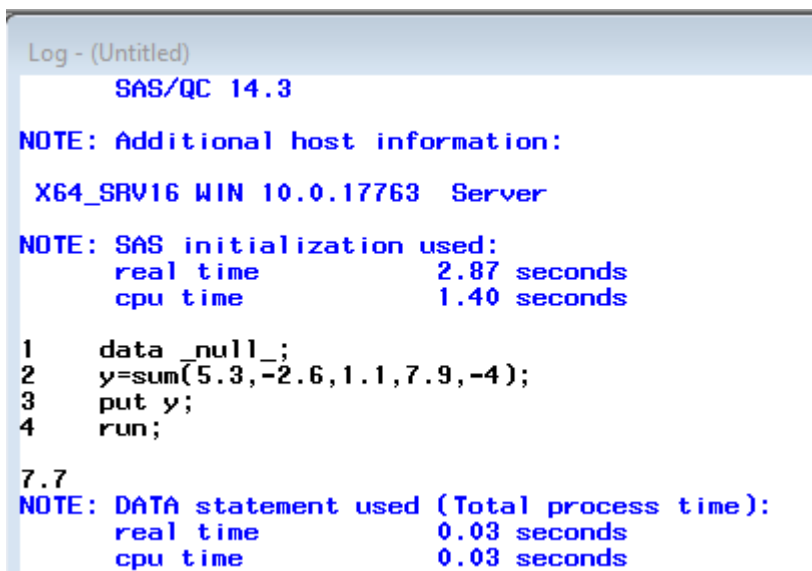
```
FUNCTIONNAME(argument-1, argument-2...argument-k);
```

Sum

The variable x is the *argument* of the function. The argument can also be input directly into the function as well

```
data _null_;  
y=sum(5.3,-2.6,1.1,7.9,-4);  
put y;  
run;
```

Sum log



```
Log - (Untitled)  
SAS/QC 14.3  
NOTE: Additional host information:  
X64_SRV16 WIN 10.0.17763 Server  
NOTE: SAS initialization used:  
real time      2.87 seconds  
cpu time       1.40 seconds  
  
1 data _null_;  
2 y=sum(5.3,-2.6,1.1,7.9,-4);  
3 put y;  
4 run;  
  
7.7  
NOTE: DATA statement used (Total process time):  
real time      0.03 seconds  
cpu time       0.03 seconds
```

null.png

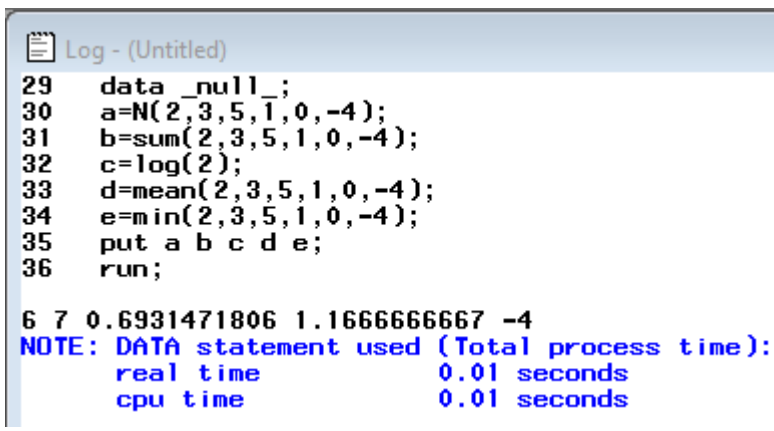
Common functions

`N(x)`: returns the number of non-missing values in `x`
`NMISS(x)`: returns the number of non-missing values in `x`
`sum(x)`: returns the sum of the elements in `x`
`mean(x)`: returns the arithmetic average of the elements in `x`
`abs(x)`: returns the absolute values of elements in `x`
`sqrt(x)`: returns the square roots of elements in `x`; is identical to `x^.5`
`log(x)`: returns the natural logarithm
`log10(x)`: returns the logarithms of base 10 (can change to desired base)

Examples of common functions

```
data _null_;
a=N(2,3,5,1,0,-4);
b=sum(2,3,5,1,0,-4);
c=log(2);
d=mean(2,3,5,1,0,-4);
e=min(2,3,5,1,0,-4);
put a b c d e;
run;
```

Common functions log



```
Log - (Untitled)
29  data _null_;
30  a=N(2,3,5,1,0,-4);
31  b=sum(2,3,5,1,0,-4);
32  c=log(2);
33  d=mean(2,3,5,1,0,-4);
34  e=min(2,3,5,1,0,-4);
35  put a b c d e;
36  run;

6 7 0.6931471806 1.1666666667 -4
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Trig (and there was much rejoicing...yay)

You can use the SAS functions freely with the assignment statement for storage and future use, use SAS functions or SAS calculations as arguments in other functions.

Pythagoras' theorem states that the length of the hypotenuse of a right triangle is equal to the square root of the sum of the squares of the other two sides. Suppose the two lesser sides of a right triangle have lengths 12.3 and 20.5. This can be done with a DATA step to create a dataset or a `_NULL_` dataset (with results printed to the log)

$$h = \sqrt{a^2 + b^2}$$

Calculate hypotenuse (`_NULL_ DATA` step)

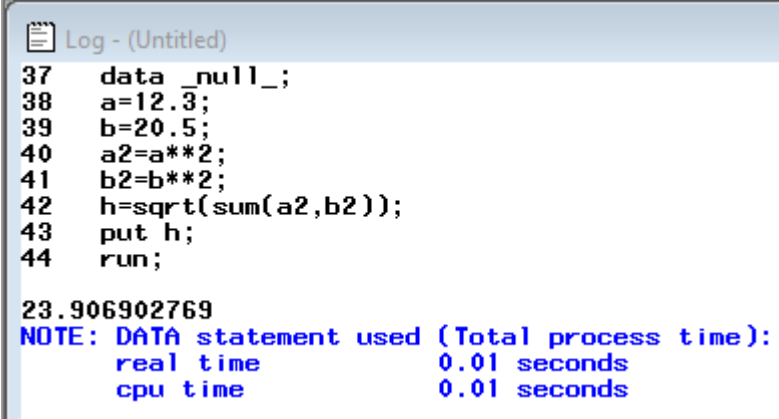
This one is done by `_NULL_ DATA` step and results are printed to the log with the PUT statement (from Module 1)

```

data _null_;
a=12.3;
b=20.5;
a2=a**2;
b2=b**2;
h=sqrt(sum(a2,b2));
put h;
run;

```

Hypotenuse log



The screenshot shows a SAS Log window titled "Log - (Untitled)". It contains the following text:

```

37 data _null_;
38 a=12.3;
39 b=20.5;
40 a2=a**2;
41 b2=b**2;
42 h=sqrt(sum(a2,b2));
43 put h;
44 run;

23.906902769
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

```

File name: null.png

Calculate hypotenuse (DATA step)

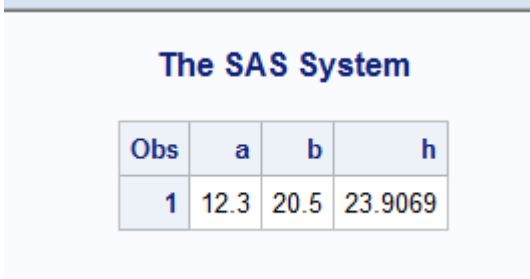
This one is done through the usual means of DATA step to create a dataset

```

data pt;
a=12.3;
b=20.5;
h=sqrt(sum(a**2,b**2));
run;
proc print data=pt;
run;

```

Hypotenuse print



The screenshot shows the output of the PROC PRINT statement. It displays a table with the following data:

The SAS System			
Obs	a	b	h
1	12.3	20.5	23.9069

File name: data.png

Creating user-defined functions in SAS

SAS also allows for user-defined functions. It is fortuitous for saving time in calculations where the same formulas are done many times. They are also nice for when a PROC does not have quite the output you are looking for

The most important concept to keep in mind whenever you write macro code is that you are writing a program that writes a program

How to build your SAS macro (user-defined function)

Macros allow you to write your own code for future use and is basically equivalent to writing your own PROC

Macro programs use two main components

- Macro statements (functions)
- Macro variables

`&name` is a macro variable reference

`%name` is macro and is called a macrocall

General form of macro definition:

```
%MACRO macro-name;  
<macro-text>  
%MEND;
```

`%MACRO`: macro creation

`Macro-name`: unique name that identifies the macro. This is named by the user and can be any name except for names of SAS statements

`macro-text`: any combination of macro statements, macro calls, text expressions, or constant text

`%MEND`: signals the end of the macro

Some predefined macros

Some macros that are predefined by SAS and you cannot use their names for your macros

- `%DO` statement
- `%DO` iterative
- `%IF` family (including `%THEN`, `%DO`, etc.)
- `%END` (not to be confused with `%MEND`)
- `%LET`: assigns a value to a macro variable

General form of `%LET`

```
%LET macro-variable-name=value;
```

`Value`: can be up to 64,000 characters

`Macro-variable-name`: is like any other SAS variable name so it can have

- 32 character maximum
- Start with letter or an underscore
- Contains only letters, underscores or numbers

Other handy options

`OPTIONS MPRINT`: has SAS print the resolved statements from the macro into the log for use in debugging. When `MPRINT` is not used, no notes are written to the log except for the macro name

Some macro variables are automatically created by the macro processor every time you start a SAS session and they can be used in programs

`&SYSDATE`: gives dates in the form of DDmmmYYYY

`&SYSDAY`: displays the day of the week

MACROS VS. MACRO VARIABLES

There are two basic building blocks of macro building: macros and macro variables. The names of macro variables start with an ampersand (&), while the names of macros start with a percent (%)

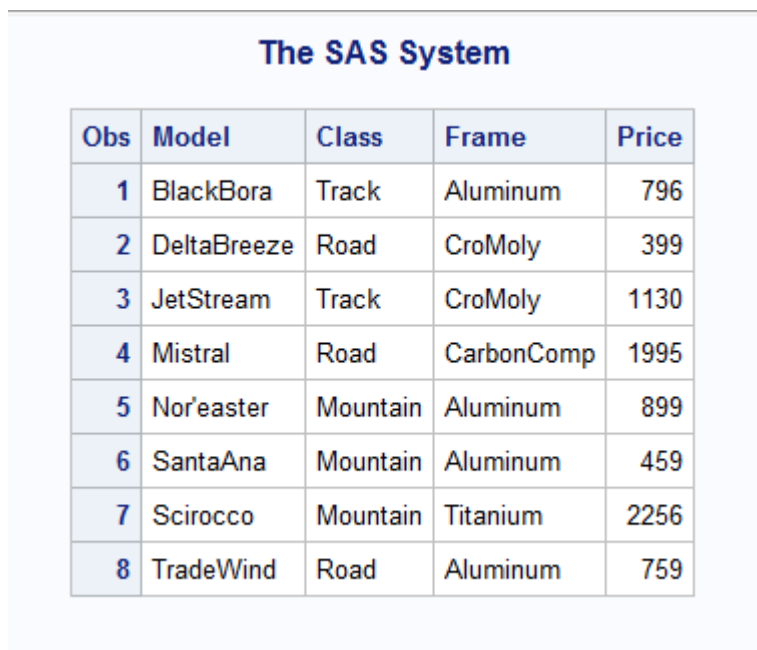
A *macro variable* is like a standard data variable except that it does not belong to a dataset and has only a single value which is always character. The value of a macro variable could be a variable name, a numeral, or any text substituted in the program

A *macro*, however, is a larger piece of a program that can contain complex logic including complete DATA and PROC steps, and macro statements such as %IF-%THEN/%ELSE and %DO-%END. Macros often, but not always, contain macro variables

Bicycle data

```
data bikemodels;
length Model $12 Class $8 Frame $15;
input Model $ Class $ Price Frame $;
cards;
BlackBora Track 796 Aluminum
DeltaBreeze Road 399 CroMoly
JetStream Track 1130 CroMoly
Mistral Road 1995 CarbonComp
Nor'easter Mountain 899 Aluminum
SantaAna Mountain 459 Aluminum
Scirocco Mountain 2256 Titanium
TradeWind Road 759 Aluminum
;
run;
proc print data=bikemodels;
run;
```

Bicycle data print



Obs	Model	Class	Frame	Price
1	BlackBora	Track	Aluminum	796
2	DeltaBreeze	Road	CroMoly	399
3	JetStream	Track	CroMoly	1130
4	Mistral	Road	CarbonComp	1995
5	Nor'easter	Mountain	Aluminum	899
6	SantaAna	Mountain	Aluminum	459
7	Scirocco	Mountain	Titanium	2256
8	TradeWind	Road	Aluminum	759

data.png

WHERE without %LET

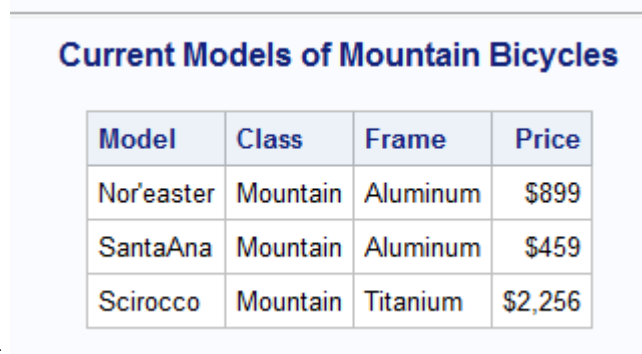
```
proc print data=bikemodels noobs;
where Class='Mountain';
format Price dollar6.;
title 'Current Models of Mountain Bicycles';
run;
title;
```

WHERE with %LET

```
%LET bikeclass=Mountain;

proc print data=bikemodels noobs;
where Class="&bikeclass";
format Price dollar6.;
title "Current Models of &bikeclass Bicycles";
run;
```

Results



Model	Class	Frame	Price
Nor'easter	Mountain	Aluminum	\$899
SantaAna	Mountain	Aluminum	\$459
Scirocco	Mountain	Titanium	\$2,256

nomac.png

Simple printing without macro

```
proc print data=bikemodels noobs;
title 'Current Models';
format Price dollar6.;
run;

proc sort data=bikemodels
  out=bikemodels2;
by Price;
run;

proc print data=bikemodels2 noobs;
title 'Current Models';
format Price dollar6.;
run;
```

Simple printing with macro

```
%macro printit;
proc print data=bikemodels noobs;
```

```
title 'Current Models';  
format Price dollar6.;  
run;  
%mend printit;  
  
%printit  
proc sort data=bikemodels;  
by Price;  
run;  
%printit
```

printit results

The image shows two SAS output windows, each containing a table of bike models. The first table is unsorted, and the second table is sorted by descending price.

Model	Class	Frame	Price
BlackBora	Track	Aluminum	\$796
DeltaBreeze	Road	CroMoly	\$399
JetStream	Track	CroMoly	\$1,130
Mistral	Road	CarbonComp	\$1,995
Nor'easter	Mountain	Aluminum	\$899
SantaAna	Mountain	Aluminum	\$459
Scirocco	Mountain	Titanium	\$2,256
TradeWind	Road	Aluminum	\$759

Model	Class	Frame	Price
DeltaBreeze	Road	CroMoly	\$399
SantaAna	Mountain	Aluminum	\$459
TradeWind	Road	Aluminum	\$759
BlackBora	Track	Aluminum	\$796
Nor'easter	Mountain	Aluminum	\$899
JetStream	Track	CroMoly	\$1,130
Mistral	Road	CarbonComp	\$1,995
Scirocco	Mountain	Titanium	\$2,256

printit.png

Sorting, printing without macro

```
proc sort data=bikemodels;
by descending Price;
run;
proc print data=bikemodels noobs;
title 'Current bikemodels';
title2 "Sorted by Descending Price";
format Price dollar6.;
```



```
run;
proc sort data=bikemodels;
by Class;
run;
proc print data=bikemodels noobs;
title 'Current bikemodels';
title2 "Sorted by Class";
format Price dollar6.;
run;
```

Sorting, printing with macro

```
%macro sortandprint(sortseq=, sortvar=);
proc sort data=bikemodels;
by &sortseq &sortvar;
run;
proc print data=bikemodels noobs;
title 'Current bikemodels';
title2 "Sorted by &sortseq &sortvar";
format Price dollar6.;
run;
%mend sortandprint;

%sortandprint(sortseq=Descending, sortvar=Price)
%sortandprint(sortseq=, sortvar=Class)
```

sortandprint macro results

**Current bikemodels
Sorted by Descending Price**

Model	Class	Frame	Price
Scirocco	Mountain	Titanium	\$2,256
Mistral	Road	CarbonComp	\$1,995
JetStream	Track	CroMoly	\$1,130
Nor'easter	Mountain	Aluminum	\$899
BlackBora	Track	Aluminum	\$796
TradeWind	Road	Aluminum	\$759
SantaAna	Mountain	Aluminum	\$459
DeltaBreeze	Road	CroMoly	\$399

**Current bikemodels
Sorted by Class**

Model	Class	Frame	Price
Scirocco	Mountain	Titanium	\$2,256
Nor'easter	Mountain	Aluminum	\$899
SantaAna	Mountain	Aluminum	\$459
Mistral	Road	CarbonComp	\$1,995
TradeWind	Road	Aluminum	\$759
DeltaBreeze	Road	CroMoly	\$399
JetStream	Track	CroMoly	\$1,130
BlackBora	Track	Aluminum	\$796

sortprint.png

Modified bicycle data

```
options mprint;  
  
data orders;  
length Model $11;  
input CustomerID $ OrderDate:date7. Model $ Quantity;  
cards;
```

```

287 15OCT03 DeltaBreeze 15
287 15OCT03 SantaAna 15
274 16OCT03 JetStream 1
174 17OCT03 SantaAna 20
174 17OCT03 Nor'easter 5
174 17OCT03 Scirocco 1
347 18OCT03 Mistral 1
287 21OCT03 DeltaBreeze 30
287 21OCT03 SantaAna 25
;
run;
proc print data=orders;
run;

```

Bicycle data 2

Current bikemodels Sorted by Class

Obs	Model	CustomerID	OrderDate	Quantity
1	DeltaBreeze	287	15993	15
2	SantaAna	287	15993	15
3	JetStream	274	15994	1
4	SantaAna	174	15995	20
5	Nor'easter	174	15995	5
6	Scirocco	174	15995	1
7	Mistral	347	15996	1
8	DeltaBreeze	287	15999	30
9	SantaAna	287	15999	25

Macro for reporting specified days with %IF-%THEN/%DO

```

%macro reports;
  %if &sysday = Wednesday %then %do;
    proc print data=orders noobs;
      format OrderDate date7.;
      title "&sysday Report: Current Orders";
    %end;
  %else %if &sysday = Friday %then %do;
    proc tabulate data=orders;
      class CustomerID;
      var Quantity;
      table CustomerID ALL, Quantity;
      title "&sysday Report: Summary of Orders";
    %end;
  %end;
%macro reports;

```

```

%end;
run;
%mend reports;
%reports

```

Report print

Wednesday Report: Current Orders			
Model	CustomerID	OrderDate	Quantity
DeltaBreeze	287	15OCT03	15
SantaAna	287	15OCT03	15
JetStream	274	16OCT03	1
SantaAna	174	17OCT03	20
Nor'easter	174	17OCT03	5
Scirocco	174	17OCT03	1
Mistral	347	18OCT03	1
DeltaBreeze	287	21OCT03	30
SantaAna	287	21OCT03	25

report.png

Build one for Pythagoras

```
options mprint;
```

```

data pdata;
input a b;
cards;
2 5
;
run;

```

Pythagoras macro

```
%let indata=pdata;
```

```

%macro pyth;
data ptm;
set &indata;
a2=a**2;
b2=b**2;
h=sqrt(sum(a2,b2));
;
proc print data=ptm;
var h;

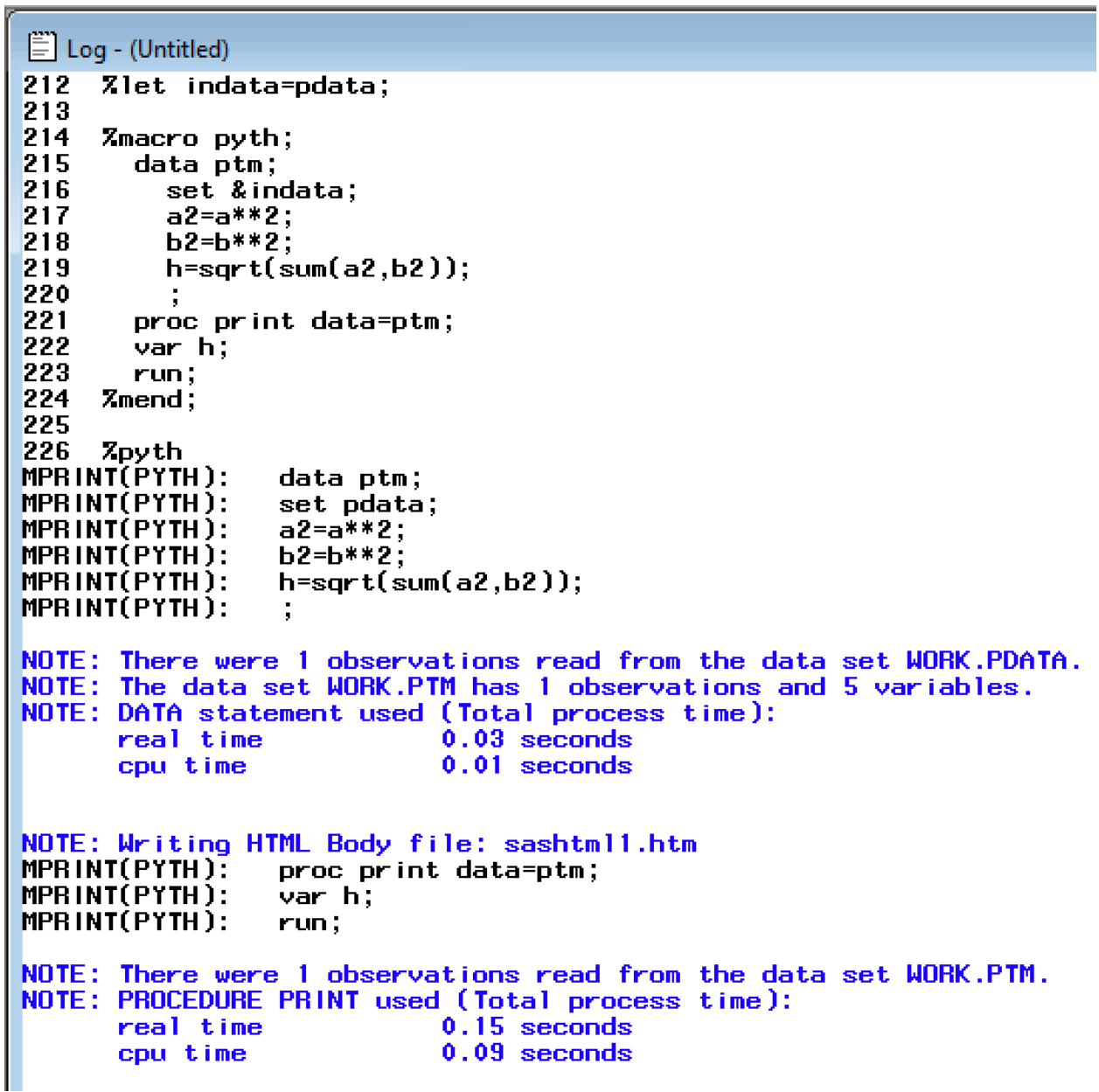
```

```
run;  
%mend;  
  
%pyth
```

Hypotenuse

Obs	h
1	5.38516

Pythagoras macro



```
Log - (Untitled)
212 %let indata=pdata;
213
214 %macro pyth;
215     data ptm;
216         set &indata;
217         a2=a**2;
218         b2=b**2;
219         h=sqrt(sum(a2,b2));
220     ;
221     proc print data=ptm;
222         var h;
223     run;
224 %mend;
225
226 %pyth
MPRINT(PYTH):    data ptm;
MPRINT(PYTH):    set pdata;
MPRINT(PYTH):    a2=a**2;
MPRINT(PYTH):    b2=b**2;
MPRINT(PYTH):    h=sqrt(sum(a2,b2));
MPRINT(PYTH):    ;

NOTE: There were 1 observations read from the data set WORK.PDATA.
NOTE: The data set WORK.PTM has 1 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds

NOTE: Writing HTML Body file: sashtml1.htm
MPRINT(PYTH):    proc print data=ptm;
MPRINT(PYTH):    var h;
MPRINT(PYTH):    run;

NOTE: There were 1 observations read from the data set WORK.PTM.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.15 seconds
      cpu time            0.09 seconds
```

macro.png

Important things of note for macros

- (1) In the macro, `x` as an argument did NOT have to be named `x`, it could be anything
- (2) The value of `x` inside the macro is “local” to the function; they exist within a special place in the computer memory set aside specifically for the macro, and their values are not available outside the function (typing `x` outside the macro will not produce any results without previously saving the value via the assignment statement in a DATA step)
- (3) Although the function `pyth` exists within the SAS environment, just like `sum` or `sqrt`, the new macro is not permanent and will disappear once the SAS session ends. Saved macros can be run again, so do not forget to save it!

Atomic Weights and Molar Masses I

A *mole* of an element or compound is defined to contain 6.02×10^{23} particles of that element or compound (atoms in the case of an element; molecules in the case of a compound). That (large) number is known as Avogadro's number. It was picked by chemists as a sort of standard amount of a substance and represents the number of atoms in 12 g of pure carbon-12. The *atomic mass* of an atom is the mass in grams of a mole of those particular atoms. A carbon-12 atom has an atomic mass of 12, and so a mole of carbon-12 atoms would weigh 12 g at sea level on Earth.

In the Earth's crust and atmosphere, some heavier or lighter isotopes of most elements occur naturally in small amounts, and so on average, a mole of everyday unpurified carbon atoms would have a mass of around 12.01 g due to trace amounts of heavier carbon (specifically carbon-14).

Atomic Weights and Molar Masses II

We want to calculate the mass of mole of a molecule of water. To do so requires finding the atomic weight of one molecule of water. A water molecule has two hydrogen atoms and one oxygen atom, and we must combine their atomic weights (commonly found in chemistry tables). The atomic weight of one hydrogen molecule is 1.008 and oxygen is 16.

$$mass = \sum a * w$$

Where:

- a: number of atoms per molecule
- w: atomic weight of each atom

*H*₂*O* data

```
options mprint;
```

```
data h2o;  
input a w;  
cards;  
2 1.008  
1 16  
;
```

```
run;  
proc print data=h2o;  
run;
```

*H*₂*O* print

Obs	a	w
1	2	1.008
2	1	16.000

Molarmass macro

```
%let chemdata=h2o;
%let molecule=Water(H2O);

%macro molarmass;
  data mmm;
  set &chemdata;
  mm=sum(a*w);
  ;
  proc summary data=mmm nway;
  var mm;
  output out=_sum_ sum=;
  run;
  title "Molar mass for &molecule";
  proc print data=_sum_;
  var mm;
  run;
  title;
%mend;

%molarmass
```

Molarmass print

Molar mass for Water(H2O)

Obs	mm
1	18.016

One more use of the function

Sucrose consists of 12 carbon, 22 hydrogen, and 11 oxygen atoms ($C_{12}H_{22}O_{11}$). The atomic weight of carbon is 12.01, with hydrogen and oxygen are already known from previous example of 1.008 and 16, respectively.

```
data sucrose;
input a w;
cards;
12 12.01
22 1.008
11 16
;
run;
* no need to rerun macro if it worked before and is same SAS session;
* NEED to reset `%LET` variables
%let chemdata=sucrose;
%let molecule=Sucrose;
```


%molar mass

Sucrose molar mass

Molar mass for Sucrose

Obs	mm
1	18.016