

Supplement: The #n option

Compare the following two codes using the #n option:

```

SAS Code
data reading;
input first $ last $ age type $
#2 s1 s2
#3 s3 s4
#4 s5;
cards;
Alicia Grossman 13 c
7.8 6.5
7.2 8.0
7.9
Matthew Lee 9 D
6.5 5.9
6.8 6.0
8.1
Elizabeth Garcia 10 C
8.9 7.9
8.5 9.0
8.8
Lori Newcombe 6 D
6.7 5.6
4.9 5.2
6.1
Jose Martinez 7 d
8.9 9.5
10.0 9.7
9.0
Brian Williams 11 C
7.8 8.4
8.5 7.9
8.0
;
run;
```

SAS Code

If you look carefully at the raw data, the values are split across multiple rows (a total of 4). Here's what happens in the first code:

- I input the variables using standard fare with

```
INPUT FIRST $ LAST $ AGE TYPE $
```

I do NOT need a #n statement here because the INITIAL/ROOT position of the pointer will exactly be where I want it to be for this INPUT statement to work.

- I use #2 to access the two variables in row "two". When I say row "two", this is relative to the INITIAL/ROOT position of the pointer.
- I use #3 to access the two variables in row "three". When I say row "three", this is relative to the INITIAL/ROOT position of the pointer. It is NOT relative to the new location of the pointer based on what happened in the previous step.
- I use #4 to access the one variable in row "four". When I say row "four", again this is relative to the INITIAL/ROOT position of the pointer. It is NOT relative to the new location of the pointer based on what happened in the previous step.

See next page for comparison to another SAS code.

SAS Code

```

data reading2;
input first $ last $ age type $
#4 s5
#3 s3 s4
#2 s1 s2;
cards;
Alicia Grossman 13 c
7.8 6.5
7.2 8.0
7.9
Matthew Lee 9 D
6.5 5.9
6.8 6.0
8.1
Elizabeth Garcia 10 C
8.9 7.9
8.5 9.0
8.8
Lori Newcombe 6 D
6.7 5.6
4.9 5.2
6.1
Jose Martinez 7 d
8.9 9.5
10.0 9.7
9.0
Brian Williams 11 C
7.8 8.4
8.5 7.9
8.0
;
run;

```

SAS Code

Notice that I'm keeping the raw data in its original format. Here's what happens in the second code:

- I input the variables using standard fare with


```
INPUT FIRST $ LAST $ AGE TYPE $
```
- For the rest of the code, notice that I only change the order in which I read in variables. I read in S5, then S3 & S4, and then S1 & S2.
- To read in S5, notice I STILL USE #4 to access that one variable in row "four". When I say row "four", again this is relative to the INITIAL/ROOT position of the pointer.
- To read in S3 & S4, notice I STILL USE #3 to access the two variables in row "three". When I say row "three", this is relative to the INITIAL/ROOT position of the pointer. It is NOT relative to the new location of the pointer based on what happened in the previous step.
- To read in S1 & S2, I STILL use #2 to access the two variables in row "two". When I say row "two", this is relative to the INITIAL/ROOT position of the pointer.

SAS Code

```

proc print data=reading;title"reading"; run;
proc print data=reading2;title"reading2"; run;

proc print data=reading2;
var first last age type s1 s2 s3 s4 s5;
title"reading2"; run;

```

SAS Code

- When you compare the output for the first two PROC PRINTs, you'll notice a strange order for READING2 ... this is because we read in the variables in the following order: S5 S3 S4 S1 S2.
- The last PROC PRINT will put the order of all variables in the same order as that of READING. That way, you can compare the output from the first and third PROC PRINTs to see if READING and READING2 really are identical ... and they are!