Session 2: Skips and Prefills

At the end of this lesson participants will be able to:

- Use the command ***skip*** to skip fields
- Use ***if then else*** statements to implement skip patterns
- End rosters from logic using ***endgroup***
- Automatically fill in fields using logic

Skips

In question **C01** we need to skip to **C03** if school attendance is "never attended" (1) or "don't know" (9).

To skip to a field we need to use logic. To add logic to a CSPro application click on the logic button on the toolbar to switch to the logic view. Here we can type in CSPro logic to control the program. Logic is added to a PROC (procedure) usually associated with a field on the form. To go to the PROC for a field in the logic view just click on the field in the form tree on the left while in logic view. We will add our skip to the PROC of field **C01** so click on **ATTENDED_SCHOOL** in the form tree.

To skip to a field we use the command ***skip***. In order to skip only when the attended school is 1 we need to combine the skip with an if statement as follows:

```
PROC ATTENDED_SCHOOL

if ATTENDED_SCHOOL = 1 then
    skip to LITERATE;
endif
```

The ***if*** statement only executes the code between the ***then*** and the ***endif*** if the condition (ATTENDED_SCHOOL = 1) is true otherwise is goes straight to the first command after the ***endif***.

This will skip to **C03** for individuals whose school attendance is "never attended". Now we need the same logic for individuals whose school attendance is "don't know".

```
PROC ATTENDED_SCHOOL

if ATTENDED_SCHOOL = 1 then
    skip to LITERATE;
endif;

if ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

Note that in CSPro, **logic statements must be separated by semicolons (;)**. This tells CSPro when one command ends and the next begins. Forgetting to put the semicolon at the end of a statement is a very common error that new users make. If you forget the semicolon, usually CSPro will tell you that that a semicolon was expected although sometimes it gets confused and gives you a less informative error message.

We can simplify this code by combining the two if statements using the ***or*** operator to create a compound condition.

```
PROC ATTENDED_SCHOOL

if ATTENDED_SCHOOL = 1 or ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

In addition to the ***or*** operator you can also make compound conditions using the *and* operator. A compound statement linked by ***or*** is true if either one of the conditions are true but a compound statement using ***and*** is only true if both conditions are true.

Finally, we should add a comment to other readers of our logic understand what we are trying to do.

```
PROC ATTENDED_SCHOOL

// Skip highest grade if never attended or don't know
if ATTENDED_SCHOOL = 1 or ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

Everything on a line after // is considered a comment and is ignored by CSPro. You can also use {} for multiline comments. It is good practice to use plenty of comments to document your logic.

Note that once we have skipped the field **C02** we can't get to it by clicking on it or going back from the following field. CSEntry keeps track of the fact that this field was skipped and won't let us back in until we change the value of **C01**. This is an important difference between ***system controlled*** and ***operator controlled*** data entry mode.

As another example lets skip the monthly rent (**F06**) if the house is not rented. Instead of comparing using "=" we can use "<>" which in CSPro logic means "not equal to".

```
PROC TENURE

// Skip monthly rent if house not rented
if TENURE <> 3 then
    skip to ROOFING_MATERIAL;
endif;
```

A note on coding style

When writing if statements, your code will be more readable and easier for others to understand if you **indent the statements between the *then* and the *endif***. It is also helpful to use consistent capitalization. We recommend all uppercase for dictionary variables like NUMBER_OF_HOUSEHOLD_MEMBERS and all lowercase for CSPro keywords like *if, then, errmsg and reenter*. Finally, you should use comments as much as possible to help others, and your future self, better understand your code.

Now let's add the skip after **Line number of father (B15)** to skip to the next person for household members under 10 years old. In this case, what do we skip to? We can't skip to **B1** (line number) since that will try to send us backwards to the line number for the current row. Instead, we use ***skip to next*** which automatically skips to the first field in the next row of the roster.

```
PROC FATHER_LINE_NUMBER

// Skip to next person for household members under 10 years of age
if AGE < 10 then
    skip to next;
endif;
```

Group exercise

Implement the skip pattern for questions **F09** (has a toilet) and **F10** (type of toilet). If the answer to question F09, have toilet, is "no", skip question F10, type of toilet. Make sure to add a comment to your code and to format your code using correct indentation and capitalization.

Other (specify) fields

We can use skips to implement other (specify) fields. Let's add the other (specify) field and skip pattern for question **F07**, roofing material. First, we add an alpha field to the dictionary to store the "***other"*** value. We can call it ROOFING_MATERIAL_OTHER and drop it onto the form so that it comes right after F06. In the PROC for F07, roofing material, we can now skip over the "***other"*** field if the response is anything other than "other" (code 5).

```
PROC ROOFING_MATERIAL

// Skip roofing material other if other code is not picked
if ROOFING_MATERIAL <> 5 then
    skip to WALL_MATERIAL;
endif;
```

Preproc and Postproc

So far, all of our examples have used **postproc** logic. **Postproc** logic is executed after the data are entered. We can also add logic to the **Preproc**. Preproc logic is executed before the data are entered.

Let's fill in the line number automatically so that the interviewer doesn't have to enter it. We can do this in the **preproc** of PERSON_NUMBER:

```
PROC PERSON_NUMBER
preproc

// Fill in line number automatically
PERSON_NUMBER = curocc();
```

The CSPro function ***curocc()*** gives us the current occurrence number of a repeating record or item. In other words, it gives the row number of the roster we are currently on.

To avoid having to hit enter to move through this field we can use the statement ***noinput*** which accepts the assigned answer and automatically moves to the next field as if the interviewer had used the enter key.

```
PROC PERSON_NUMBER
preproc

// Fill in line number automatically
PERSON_NUMBER = curocc();
noinput;
```

Alternatively, we can make the field uneditable by checking the protected box in the field properties. With noinput it is still possible to go back and modify the field but protected fields cannot be modified except from logic. Be aware that if you do not set the value of a protected field in logic before the interviewer gets to the field, CSEntry will give you an error.

Group exercise

In question B9, the year that the person first moved to this district the instruction says:

> "For respondents enumerated in their district of birth enter year and month of birth."

Implement this by testing if the district field in section A matches the district of birth in **B07**. If they match, set the month and year moved to this district in question **B09** to the month and year of birth in question **B06** and do not ask question **B09**. You may be tempted to use the skip statement for this but you should consider using the ***noinput*** statement instead.

Endgroup

Instead of using the occurrence control field in the roster, we could ask the user if they want to terminate the roster. To do that, we first add a new field to the dictionary and it to the roster. Let's call it **MORE_PEOPLE** and give it the value set Yes - 1, No – 2. We will put it at the end of the names roster. If the interviewer picks "no" then we use the command ***endgroup*** which terminates entry of the current roster or repeating form.

```
PROC MORE_PEOPLE

// Exit roster when no more people in household
if MORE_PEOPLE = 2 then
    endgroup;
endif;
```

With this, we no longer need to use the occurrence control field of the roster. However, since the other rosters (demographics and education) still rely on the variable NUMBER_OF_HOUSEHOLD_MEMBERS as a roster control field we need to set its value after completing the names roster. We can make it a protected field, move it after the names roster and set it using logic:

```
PROC NUMBER_OF_HOUSEHOLD_MEMBERS
preproc
// Set number of household members to size of person roster.
// It is used as roster occurrence control field for later rosters.
NUMBER_OF_HOUSEHOLD_MEMBERS = totocc(NAMES_ROSTER);
```

This uses the function ***totocc()*** which gives the total occurrences of a repeating record or item. In other words, it gives us the total number of rows of a roster.

Endlevel

There is also the ***endlevel*** command which is similar to ***endgroup*** but skips the rest of the current questionnaire (except for two level applications when in a second level node where it terminates the current node).

Referring to Variables in Other Rosters

We are only supposed to fill in the education section (section C) for those aged three years and above. We can use skip to skip over the education questions if the age is less than three. We need to do this in the **preproc** of C01, the first field in the education roster, **ATTENDED_SCHOOL**.

Note that since this field already has a **postproc** we need to add the **preproc** first and then add the word "postproc" so that the original **postproc** logic does not become part of the **preproc**.

```
PROC ATTENDED_SCHOOL
preproc

// Skip education for household members under 3 years of age
if AGE < 3 then
    skip to next;
endif;

postproc
// Skip highest grade if never attended or don't know
if ATTENDED_SCHOOL = 1 or ATTENDED_SCHOOL = 9 then
    skip to LITERATE;
endif;
```

When we run this it doesn't work. The problem is that **AGE** is in a different roster from **ATTENDED_SCHOOL**. Remember that there are multiple copies of the variable AGE: one for each member in the household. We can refer to AGE for specific members of the household using a subscript. For any item that is repeated, adding a number in parentheses after it gives you the value of a particular occurrence of the variable. For example AGE(1) will be the age of the first household member, AGE(2) the age of the second member… If we omit the subscript when executing logic in a proc of a roster CSPro will assume that we want the one for the current row of the current roster. However, in this case we are not in the section B roster so we need to specify the subscript. Since there is a one to one correspondence between the rows of the section C roster and the rows of the section B roster we can use the current row number in the section C roster as the subscript.

```
PROC ATTENDED_SCHOOL
preproc

// Skip education for household members under 3 years of age
if AGE(curocc()) < 3 then
    skip to next;
endif;
```

We will need to use subscripts anytime we refer to a variable in a roster different from the one that we are currently in.

Skipping Based on Checkboxes

We only want to ask question **B11**, sign language, if the household member is hearing disabled. We want to skip question **B11** if hearing disabled is not checked in question **B10**. How can we tell in logic if hearing disabled is checked? Hearing disabled is option B in the value set so we need to determine if the set of all checked options contains the letter B.

In CSPro logic we can use the function ***pos()*** which returns the position of one string within another. For example ***pos("C", "CSPro")*** will be one, ***pos("P", "CSPro")*** will be three and ***pos("o", "CSPro")*** will be five. If the search string is not found, ***pos()*** will return zero. In our case, the letter B is not always at the same position in the string. If the interviewer chooses just option B then B will be the first character in the string but if the interviewer chooses A and B then the result will be "AB" and B will be in position two. However, all we care about is whether or not B is in the result string. For this we can check the result of pos(). If the result of pos() is nonzero then B is in the string and if the result is zero, B is not in the string.

Using *pos()* the logic for skipping sign language if hearing is not checked is:

```
PROC DISABILITIES

// Skip sign language unless hearing disabled (option b) is checked
if pos("B", DISABILITIES) = 0 then
    skip to MOTHER_ALIVE;
endif;
```

Exercises

1. Skip question B08 (residence one year ago) if age is less than 1-year-old.
2. Skip question B17 (age at first marriage) if marital status is not married, divorced, or widowed.
3. In section E, deaths, implement the skip pattern for question E01, any deaths.
4. In section E, deaths, implement the skip pattern for question E09, died while pregnant.
5. In section E, deaths, skip questions E09 and E10 for household members who are NOT women aged 12-50.
6. In section E, use logic to pre-fill the line number field. Make sure that the line number is protected.
7. Add an additional question to F12 to determine if the respondent wants to give the distance to water in minutes or kilometers. If they choose kilometers then skip the distance in minutes question otherwise skip the distance in km question.
8. Add the other(specify) field and skip pattern for F08, wall material.
9. Add the other(specify) field for B19, languages spoken.
10. In question G01, household possessions, skip the value field if the quantity is zero.
11. Add variables and a form for section D: Fertility. Add the variables to the person record but create a new form with its own roster just like we did for section C. Don't worry about the last question on the form that displays the total births. We will cover that in a later lesson. Add the skip patterns for the fertility section. Don't forget to skip the entire section for males and females NOT between 12 and 50 years old.