Session 2: Skips and consistency checks

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

- Use the commands ***skip*** and **ask if** to skip fields
- Use ***if then else*** statements to implement skip patterns
- End rosters from logic using ***endgroup***
- Automatically fill in fields using logic
- Implement consistency checks using multiple variables

Skips

In question **PXYCHK** we need to skip the rest of the proxy section if they answer "no". 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 **PXYCHK** so click on **PXYCHK** in the form tree.

To skip to a field, we use the command ***skip***. In order to skip only when the answer is no, we need to combine the skip with an if statement as follows:

```
PROC PXYCHK

if PXYCHK = 2 then
    skip to EMPPAY;
endif
```

The ***if*** statement only executes the code between the ***then*** and the ***endif*** if the condition (PXPYCHK = 2) is true otherwise it goes straight to the first command after the ***endif***.

This will skip to **EMPPAY** for individuals who answer "no" to **PXYCHK**.

In addition to "=" to check if two values are equal, you can use the following operators:

Operation	Symbol
Equal to	=
Not equal to	<>
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=

For example, to implement the universe of age 5 and above for the highest level of education (**DEMEDL**) we could add the following logic to the postproc of the preceding question.

```
PROC DEMAG1
if DEMAG1 < 5 then
    skip to DEMEDC;
endif
```

Ask if

An alternative to skip is the statement ***ask if***. It skips the current question if the condition is NOT met. For example, to use this ask if order to only ask the question for those 5 and over we can use the following logic:

```
PROC DEMEDL
preproc
ask if DEMAG1 >= 5;
```

The ***ask if*** statement skips over the question *unless* the condition DEMAG1 >= 5 is true. Note that when changing from skip to ask, the condition was negated by switching from *"<"* to *">="*.

The line ***preproc*** tells CSPro to execute this logic before asking the question. Our previous examples did not specify **preproc**, so by default the logic was executed in the **postproc**. **Postproc** logic is executed after the data are entered. **Preproc** logic is executed before the data are entered. For each variable, you can specify a preproc, a postproc or both.

We also need to skip currently attending school (**DEMEDC**) for those who are not between 5 and 29 years of age. We can use **ask if** for this as well:

```
PROC DEMEDC
preproc
ask if DEMAG1 >= 5;
ask if DEMAG1 <= 29;
```

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.

The two ask ifs can be simplified to a single ask-if with by linking the two conditions together with *"and"*:

```
PROC DEMEDC
preproc
ask if DEMAG1 >= 5 and DEMAG1 <= 29;
```

An *"and"* expression is true when both the left and right expressions are true so the question will only be asked if the age is both greater than or equal to 5 *and* less than or equal to 29.

In addition to *"and"*, CSPro supports the following logical operators:

Operation	Keyword
Negate an expression	not
True if both expressions are true	and
True if either expression is true	or

We can make our expression even simpler using the **in** operator which checks if a value is in a range:

```
PROC DEMEDC
preproc
ask if DEMAG1 in 5:29;
```

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

```
PROC DEMEDC
preproc
// Only ask highest grade if age is between 5 and 29.
ask if DEMAG1 in 5:29;
```

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 **DEMEDC** 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 **DEMAG1**. This is an important difference between ***system controlled*** and ***operator controlled*** data entry mode.

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.

Group exercise

Implement the skip pattern for question **ABSWHY** (why did you not work last week). Only ask **ABSWHY** if the previous question, **ABSJOB**, is "yes". 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 ask if to implement other (specify) fields. Let's add the other (specify) field and skip pattern for the question **ABSWHY**. First, we add an alpha field to the dictionary to store the "*other*" value. We can call it **ABSWHY_OTHER** and drop it onto the form so that it comes right after **ABSWHY**. In the PROC for **ABSWHY_OTHER** we can use **ask if** to skip the question if the other code (13) is not chosen in **ABSWHY**.

```
PROC ABSWHY_OTHER
preproc

// Only ask other question if other code is picked
ask if ABSWHY = 13;
```

Skip to next

Now let's add the skip for the module universe for the proxy module. We should skip the entire module for all household members under 15. In fact, we want to skip all the remaining modules and start the proxy module for the next household member. We can add a skip in the preproc of the first field of the proxy module (**PXYRSP**) but what do we skip to? We can't skip to **PXYRSP** since that is the field we are already on. Instead, we use ***skip to next*** which automatically skips to the first field in the next row of the roster.

```
PROC PXYRSP
preproc

// Skip to next person for household members under 15 years of age
if DEMAG1 < 15 then
    skip to next;
endif;
```

Subscripts

There is a small problem with this. Since **DEMAG1** is in the household roster, we are not necessarily referring to the age of the correct person when we reference it from **PXYRSP**. If the order of the household members in the labor roster is different from the order in the demographics roster when we reference age we could be getting the age of a different household member. Remember that there are multiple copies of the variable DEMAG1: one for each member in the household. We can refer to DEMAG1 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, DEMAG1(1) will be the age of the first household member, DEMAG1(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 demographics roster so we need to specify the subscript.

Before we can add a subscript, we need to a variable to the labor record to link it back to the demographic record. Add a new variable to the labor record named **LBPPNO** in which the interviewer will enter the line number from the demographics roster for the person currently being interviewed. Put that variable on the form at the start of the proxy section. Now we can use it as a subscript in our skip pattern.

```
PROC PXYRSP
preproc

// Skip to next person for household members under 15 years of age
if DEMAG1(LBPPNO) < 15 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.

Prefills

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 **PPNO**:

```
PROC PPNO
preproc

// Fill in line number automatically
PPNO = 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 PPNO
preproc

// Fill in line number automatically
PPNO = 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 a catastrophic error.

An even easier way to set the value in this case is to check the **sequential** box in the field properties to have CSEntry automatically set the value for you. With sequential we do not to add any logic to the field.

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 **DEMMORE** and give it the value set Yes – 1, No – 2. We will put it at the end of the demographic roster. If the interviewer picks "no" then we use the command ***endgroup*** which terminates entry of the current roster or repeating form.

```
PROC DEMMORE

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

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

```
PROC DEMNMEM
preproc
// Set number of household members to size of demographic roster.
// It is used as roster occurrence control field for later rosters.
DEMNMEM = totocc(DEM_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).

Consistency Checks with Two Variables

Up to now, we have been able to limit the responses for each variable to only a set of valid responses using value sets. What if we want to check for consistency between two different variables?

For example, let's prevent the interviewer from entering a head of household under 15 years old. Click on the DEMAG1 field in the forms tree and then click on *Logic* in the toolbar to open the logic editor. Add the following code in the procedure for DEMAG1:

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("Head of household must be at least 15 years old");
    reenter;
endif;
```

The ***if*** statement only executes the code between the ***then*** and then ***endif*** if the condition (DEMREL = 1 and DEMAG1 < 15) is true. The ***errmsg*** statement will display a message to the user. The ***reenter*** statement forces the interviewer to stay in the current field and does not allow them to advance to the next field.

Note that we put our logic in the proc for DEMAG1 and not in the proc for DEMREL. This is because when we are in the proc for DEMREL, the value for DEMAG1 has not yet been entered. When creating consistency checks we always put the logic for the check in the proc for the *last* field involved in the check.

Better Error Messages

We can improve our error message by adding parameters to the string. Let's display the age in the error message:

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("Head of household is only %d years old. The head of household
must be at least 15 years old", DEMAG1);
    reenter;
endif;
```

The "**%d**" in the message is replaced by the value of the variable that follows. "**%d**" is used for numeric variables and "**%s**" is used for alpha variables.

It would be better if we could include the name of the person in the error message as an additional clue to the interviewer. We can add a second parameter for the name. Since the name is an alpha variable we use **%s** instead of **%d.** *(%d is for decimal values).* If you have multiple parameters in a message, the order of the variables after the message must match the order of the parameters in the message so we must put NAME first, before DEMAG1.

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("%s is the head of household and is only %d years old. The head of
household must be at least 15 years old", NAME, DEMAG1);
    reenter;
endif;
```

Why is there a whole bunch of extra space after the name in the error message? Remember that alpha variables in the dictionary are fixed length. This means that they get padded with blank spaces. We can remove the trailing blank spaces by using the ***strip()*** function.

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("%s is the head of household and is only %d years old. The head of
household must be at least 15 years old", strip(NAME), DEMAG1);
    reenter;
endif;
```

Errmsg with select

If we add select to the ***errmsg*** we can give the user the option of going back to correct either field.

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("%s is the head of household and is only %d years old. The head of
household must be at least 15 years old", strip(NAME), DEMAG1)
    select("Fix relationship", DEMREL,
           "Fix age", DEMAG1);
endif;
```

With the ***select*** we no longer need the ***reenter*** since CSEntry will automatically reenter the field that the interviewer selects.

Note that the ***select*** clause is part of the ***errmsg*** statement so there is no semicolon in between the ***select*** and the ***errmsg***.

If you want to allow the user to ignore the error and enter the next field, you can add an option to the select statement with the keyword ***continue***:

```
PROC DEMAG1

// Ensure that the head of household is at least 15 years old.
if DEMREL = 1 and DEMAG1 < 15 then
    errmsg("%s is the head of household and is only %d years old. The head of
household must be at least 15 years old", strip(NAME), DEMAG1)
    select("Fix relationship", DEMREL,
           "Fix age", DEMAG1);
           "Ignore error", continue);
endif;
```

Now the message dialog will have a third button labelled “ignore” that, when clicked, will move to the next field, in this case to level of education.

This is commonly referred to as a “soft edit check” as opposed to the previous “hard edit check” that does not allow the interviewer to move on until the inconsistency is fixed. The advantage of a soft edit check is that the interviewer won’t get stuck, however, data quality may suffer. For soft edit checks, CSPro also provides a **warning** function, which is identical to **errmsg** except that if you have already ignored the message, when you navigate through the field by clicking on the case tree or resuming from partial save, the message is not shown a second time.

Testing Skips and Consistency Checks

When testing consistency checks involving multiple variables it is important to test all possible combinations of the variables. To do this we can create a test matrix. For example, to test all possible combinations of our consistency check between DEMAG1 and DEMREL we would use the following matrix that shows the expected result for each combination of the variables involved:

	DEMREL	
DEMAG1	Head (1)	Not head (> 1)
< 15	Error	OK
>= 15	OK	OK

Using this matrix, you can test each of the four possible combinations and make sure that you get the expected result. This ensures that all possible cases are tested.

By combining the test matrices from all the consistency checks in the application, you can create a test plan for the survey/census application to verify the application works correctly when changes are made. Often times changes to one part of the questionnaire can have unintended effects on skips and consistency checks in other parts of the questionnaire so it is important to have a test plan that is used after every set of changes to ensure that no problems are introduced.

Exercises

1. Use skip or ask if to skip over questions EMPPFT and EMPFAM if the answer to EMPAY is “No”.
2. Implement the module universe for the temporary absence module.
3. Implement the skip patterns/universes for the temporary absence module according to the specifications. Note that you can use the “in” operator with a list of values (e.g. “*if ABSWHY in 2,7,8”*) which may help with ABSDUR and ABSPAY. For any routing in the specification that go to the agriculture (AGF) module that we have not yet implemented, simply use skip to next to move to the next row of the roster. We will correct this once we add the missing module.
4. Add a consistency check to make sure that if the highest level of education (DEMEDL) is tertiary, the age must be 18 or above. Use errmsg with select to show a message and buttons to allow the interviewer to correct either age or level of education.
5. Add a consistency check to ensure that the relationship of the household member in the first row of the demographics roster is the head of household (DEMREL = 1). Hint: remember the curocc() function?
6. If the date of birth (DEMBTH) is known, automatically fill in the age (DEMAG1). The age is known if the year is not 9999, and the month and day are not 99. You can use the subitems to look at the values of day, month and year separately. You can calculate the age by taking the difference of the interview date and the age. CSPro has a function datediff() that can do this calculation for you. Search for “datediff” in the CSPro help for details. Make sure you DO NOT skip the age field when you fill it in. Instead, use “noinput” if the date of birth is known so that the user does not have to enter the age. Finally, you will have to change the errmsg select statements that route to the age field to instead go to the date of birth.