Logic Flow - Conditional Processing

Objectives

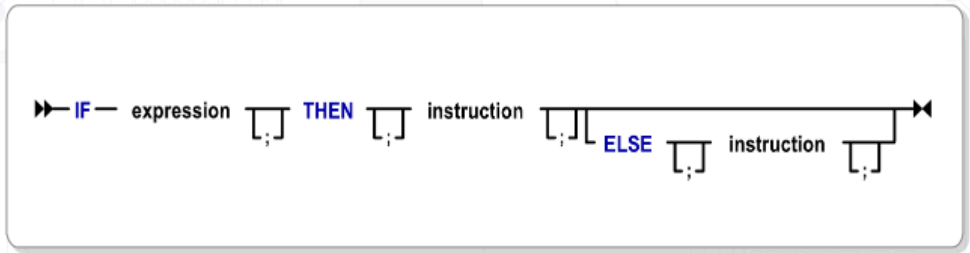
Logic Flow - Conditional Processing

In this module, you will examine the two types of conditional instructions that REXX supports.

You will also discover how conditional instructions enable a program to choose different paths depending on whether specific conditions are met or not.

After completing this module, you will be able to:

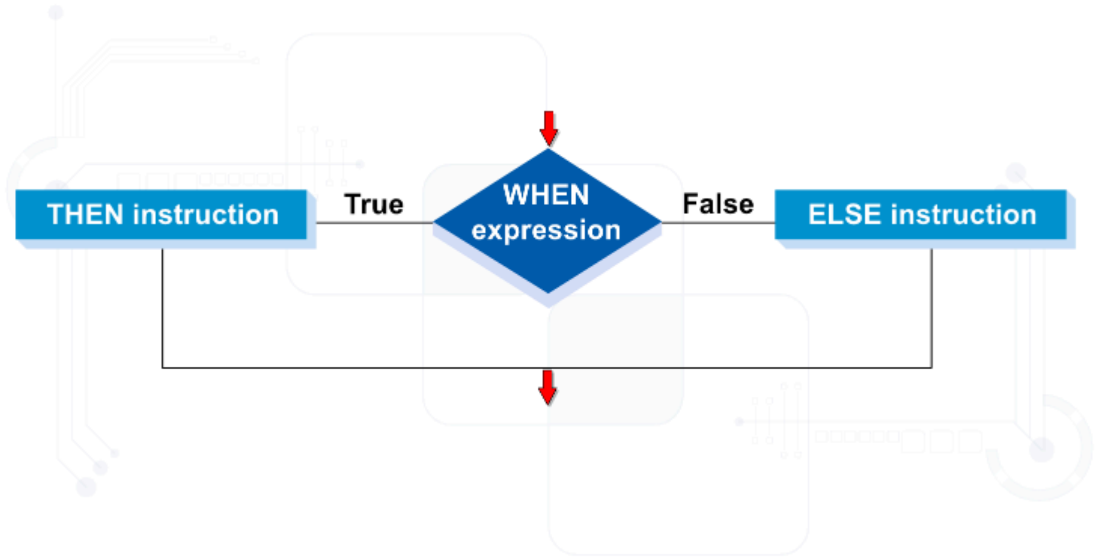
- Recognize the IF/THEN/ELSE Construct Keyword Instructions
- Recognize the SELECT/WHEN/OTHERWISE Construct Keyword Instructions



Conditional processing enables the logic of a program to take different paths and perform different actions depending on whether specific conditions are met or not.

The `IF` instruction evaluates an expression. If the expression is true, the following instruction `THEN` is executed.

Optionally, the `ELSE` clause enables a different instruction to be executed if the expression evaluates to false.

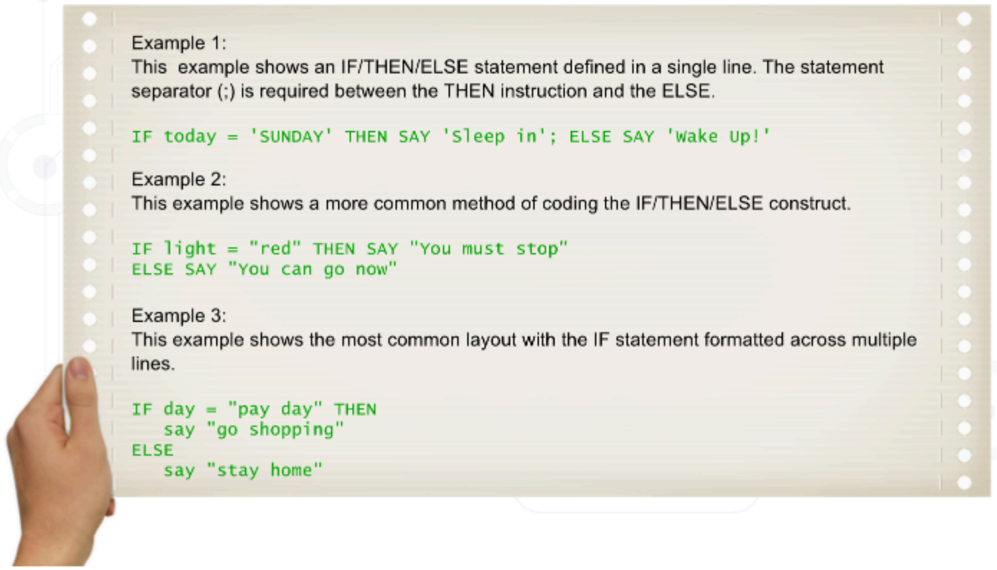


This flowchart illustrates how the logic of the IF/THEN/ELSE construct works. This can be coded as:

```
IF expression THEN instruction /* true */
ELSE instruction /* false */
```

Alternatively, the instruction to be executed can be coded immediately after the branch clause like this:

```
IF expression THEN
instruction /* true */
ELSE
instruction /* false */
```



Example 1:
This example shows an IF/THEN/ELSE statement defined in a single line. The statement separator (;) is required between the THEN instruction and the ELSE.

```
IF today = 'SUNDAY' THEN SAY 'Sleep in'; ELSE SAY 'Wake Up!'
```

Example 2:
This example shows a more common method of coding the IF/THEN/ELSE construct.

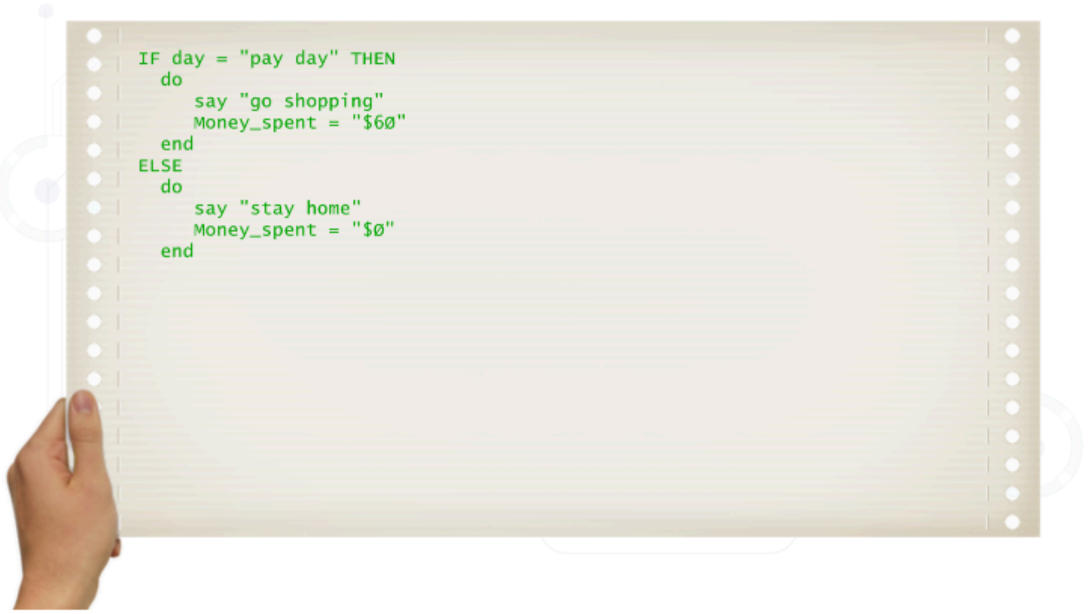
```
IF light = "red" THEN SAY "You must stop"
ELSE SAY "You can go now"
```

Example 3:
This example shows the most common layout with the IF statement formatted across multiple lines.

```
IF day = "pay day" THEN
   say "go shopping"
ELSE
   say "stay home"
```

The IF/THEN/ELSE construct can be coded in single or multiple lines.

As shown in Example 3, the instruction executed in the event of a true or false expression can be coded on the line after the THEN or the ELSE. In this case, a comma is not required to indicate a continuation.

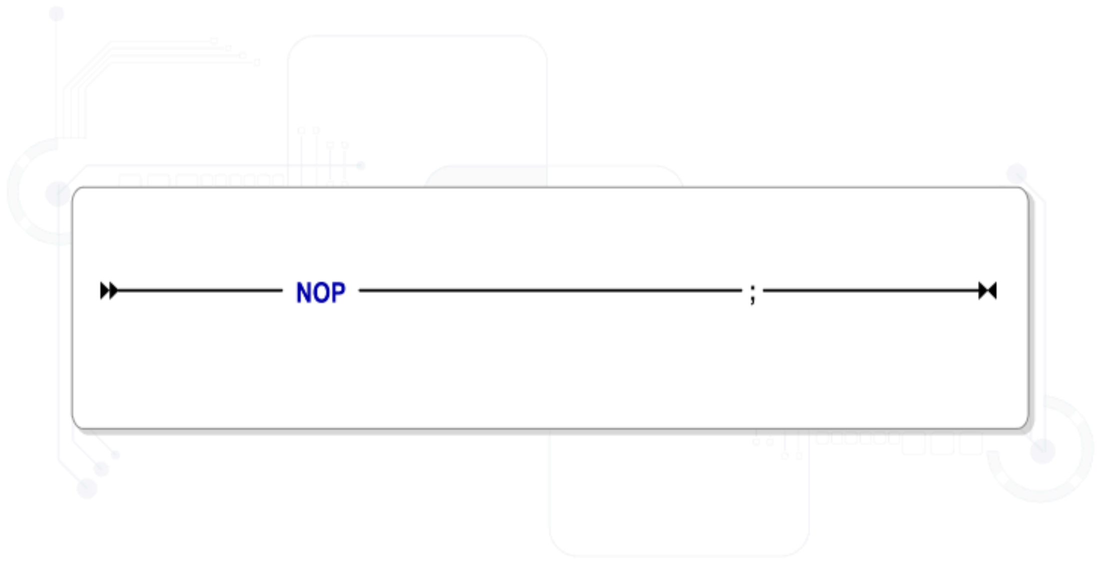


```
IF day = "pay day" THEN
  do
     say "go shopping"
     Money_spent = "$60"
  end
ELSE
  do
     say "stay home"
     Money_spent = "$0"
  end
```

THEN and ELSE clauses enable only the next instruction that is encountered to be executed.

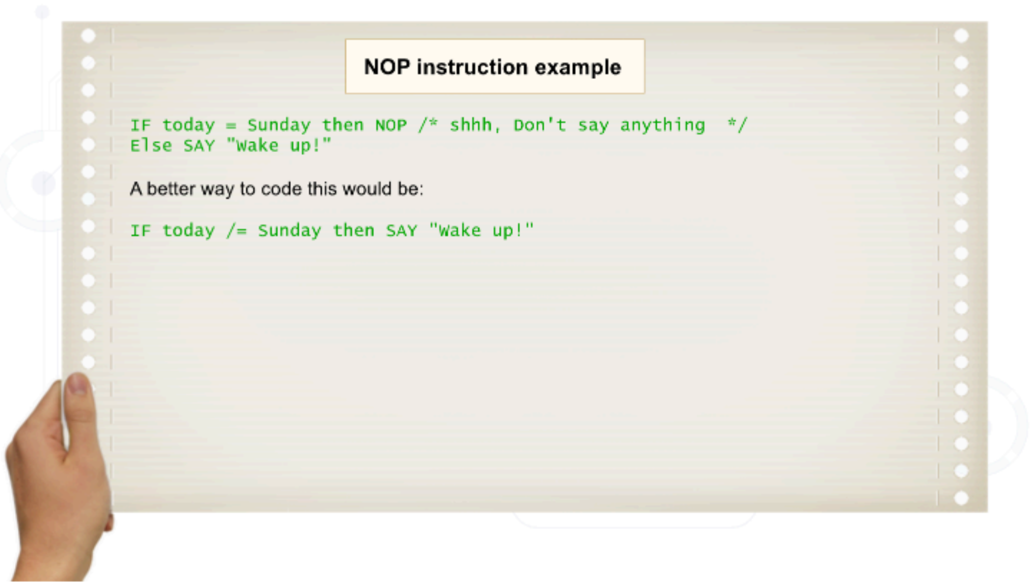
When multiple instructions are required to be executed, the DO and END keywords can be used. A group of instructions can then be executed for THEN or ELSE clauses.

Shown here is an example of the DO and END instructions being used in an IF/THEN/ELSE construct to enable multiple instructions to be executed by THEN or ELSE.



The no operation (NOP) instruction can be used in situations where a clause may require an instruction, but there is no programming requirement.

The NOP instruction does nothing.



NOP instruction example

```
IF today = Sunday then NOP /* shhh, Don't say anything  */
Else SAY "Wake up!"
```

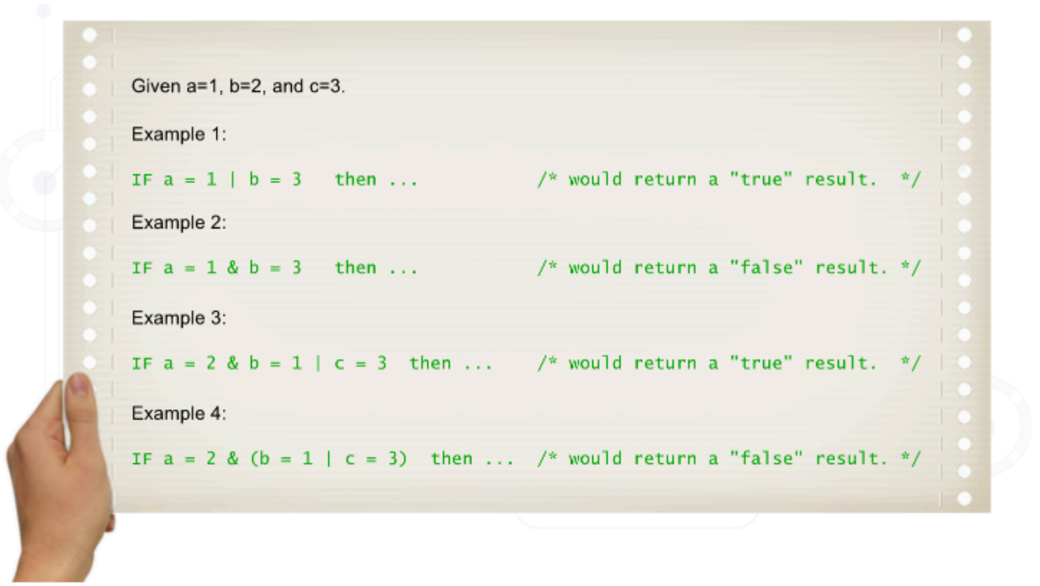
A better way to code this would be:

```
IF today /= Sunday then SAY "Wake up!"
```

Generally, at least one instruction should follow the THEN and ELSE clauses. When either clause has no instruction requirement, it is good practice to include the NOP instruction after it, as shown in this example.

It is particularly important to use NOP when nesting IF instructions because it helps to prevent bugs that can cause unexpected results.

Note that it is not mandatory to code the ELSE clause if it is not required.


Given a=1, b=2, and c=3.

Example 1:

```
IF a = 1 | b = 3   then ...           /* would return a "true" result.  */
```

Example 2:

```
IF a = 1 & b = 3   then ...           /* would return a "false" result. */
```

Example 3:

```
IF a = 2 & b = 1 | c = 3  then ...    /* would return a "true" result.  */
```

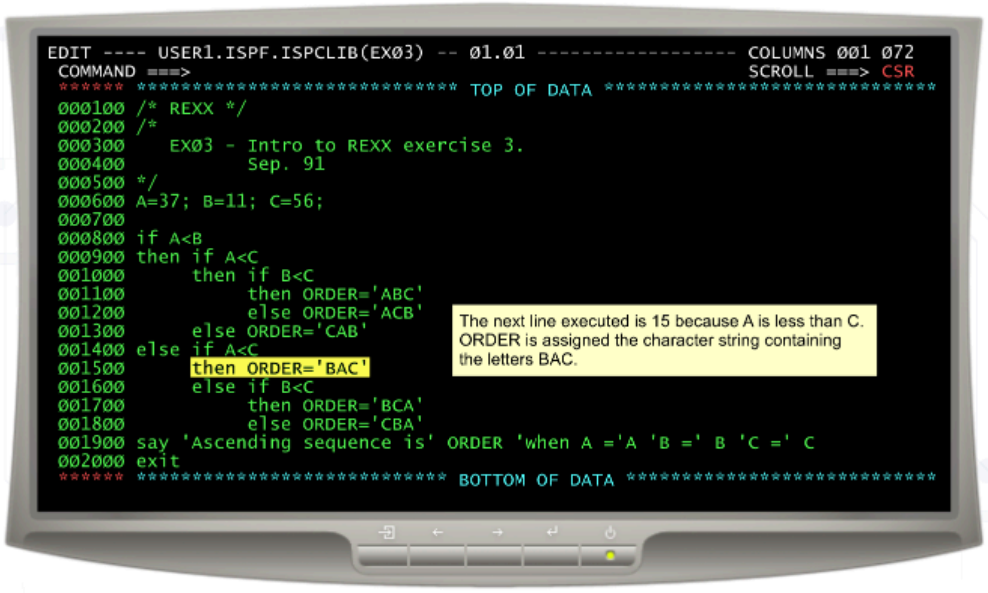
Example 4:

```
IF a = 2 & (b = 1 | c = 3)  then ... /* would return a "false" result. */
```

When evaluating IF clauses, multiple conditions can be specified by using the Boolean log characters & for "and" and | for "or".

Many expressions can be included with multiple Boolean operators, but to avoid confusion with precedence, since "and" is processed before "or", you can use parentheses to clarify the required order of precedence.

Shown here are some examples of multiple expressions being used in an IF statement.



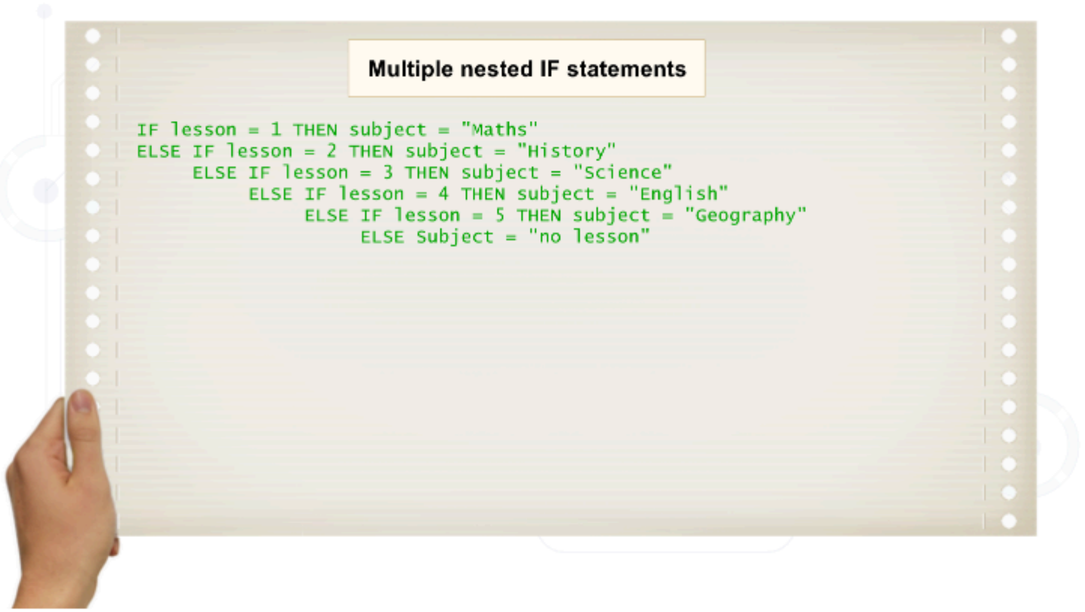
```
EDIT ---- USER1.ISPF.ISPCLIB(EX03) -- 01.01 ------------------ COLUMNS 001 072
 COMMAND ===>                                                  SCROLL ===> CSR
 ****** ***************************** TOP OF DATA ******************************
 000100 /* REXX */
 000200 /*
 000300    EX03 - Intro to REXX exercise 3.
 000400           Sep. 91
 000500 */
 000600 A=37; B=11; C=56;
 000700
 000800 if A<B
 000900 then if A<C
 001000      then if B<C
 001100           then ORDER='ABC'
 001200           else ORDER='ACB'
 001300      else ORDER='CAB'
 001400 else if A<C
 001500      then ORDER='BAC'
 001600      else if B<C
 001700           then ORDER='BCA'
 001800           else ORDER='CBA'
 001900 say 'Ascending sequence is' ORDER 'when A ='A 'B =' B 'C =' C
 002000 exit
 ****** **************************** BOTTOM OF DATA ****************************
```

The next line executed is 15 because A is less than C. ORDER is assigned the character string containing the letters BAC.

Here is an example of the IF/THEN/ELSE construct in use.

The THEN parameter of the IF statement is coded on the line following the IF statement. This format is acceptable, but it is more common to code them on the same line.

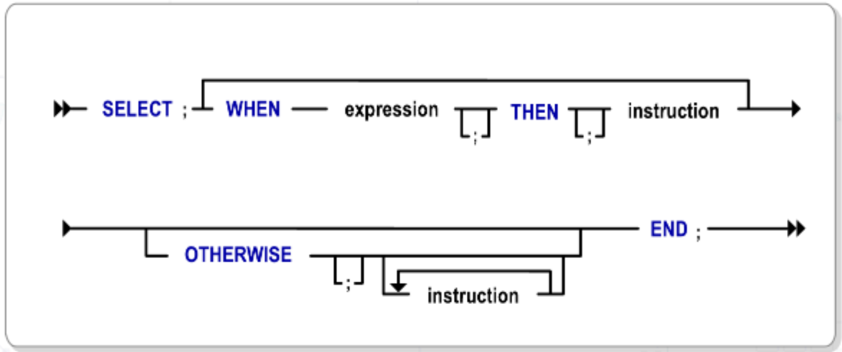
Click **Play** to see how the logic of this program works.



Multiple nested IF statements

```
IF lesson = 1 THEN subject = "Maths"
ELSE IF lesson = 2 THEN subject = "History"
     ELSE IF lesson = 3 THEN subject = "Science"
          ELSE IF lesson = 4 THEN subject = "English"
               ELSE IF lesson = 5 THEN subject = "Geography"
                    ELSE Subject = "no lesson"
```

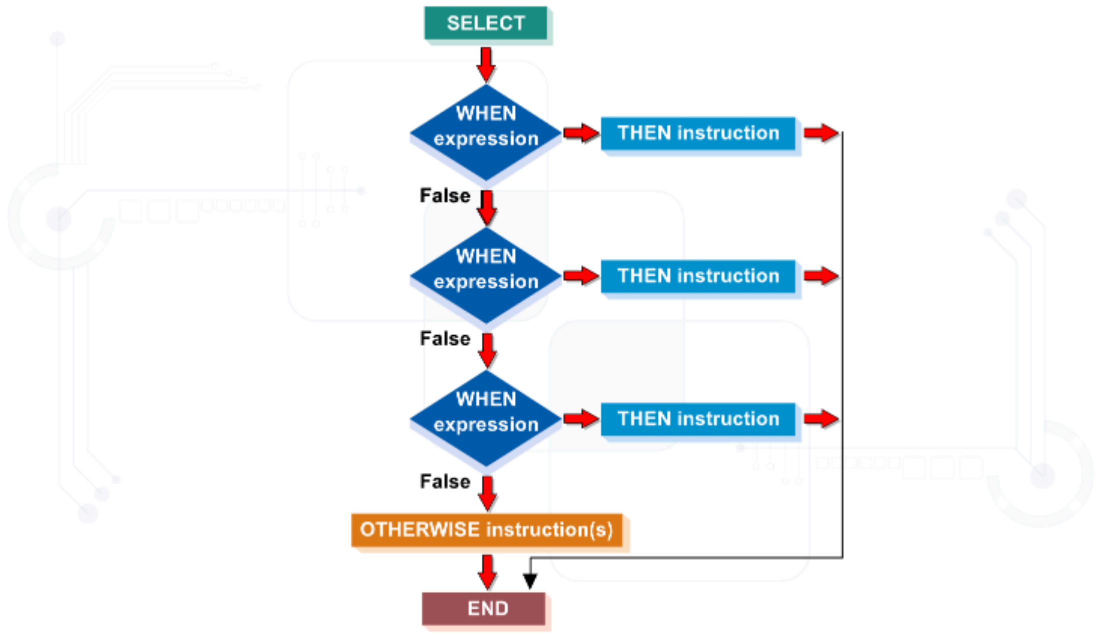
When multiple choices are available but only one is required, a solution is to code multiple nested IF statements like the example shown here.



Alternatively, the SELECT instruction enables processing to continue down only one of several possible paths. The first WHEN clause expression that evaluates true will have its THEN clause executed.

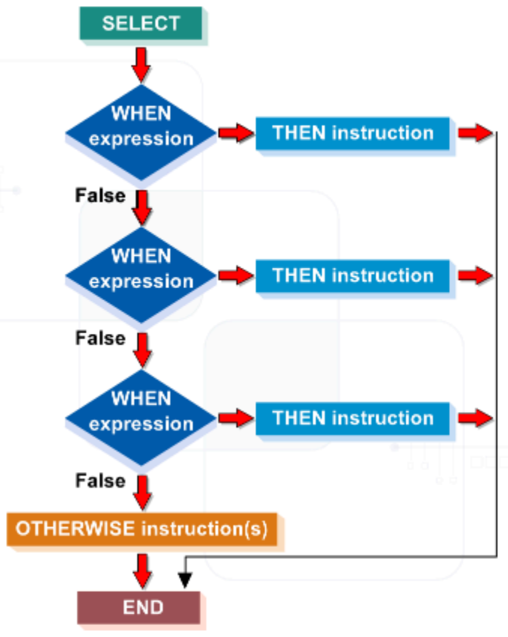
If no WHEN clauses evaluate true, the instruction following OTHERWISE is executed.

The SELECT instruction is a better and more efficient method of coding multiple options when only one will be correct.



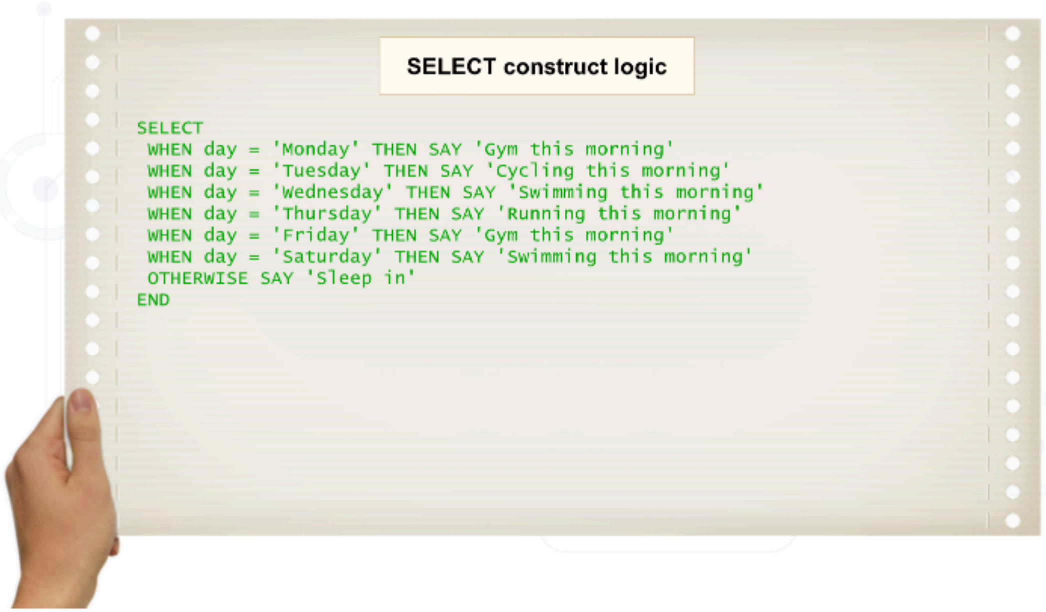
The SELECT statement can be coded as follows:

```
SELECT
WHEN expression THEN instruction
WHEN expression THEN instruction
WHEN expression THEN instruction
OTHERWISE instruction(s)
END
```



The SELECT instruction flows from top to bottom and branches to the first true expression. If no expressions evaluate true, the instructions in the OTHERWISE clause are executed.

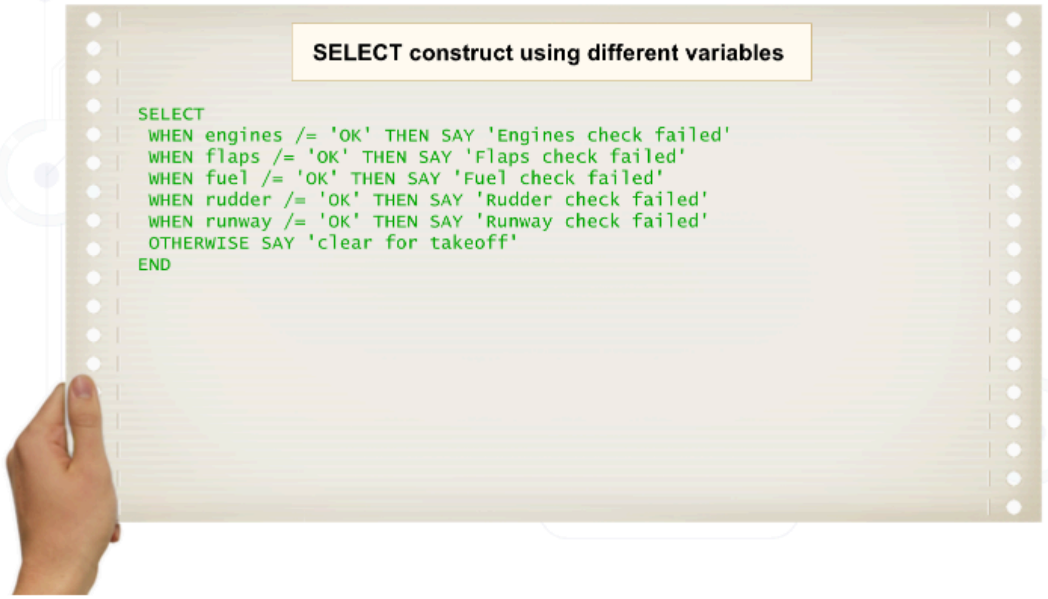
The OTHERWISE clause is not mandatory, but if no WHEN expression evaluates as true and there is no OTHERWISE clause, a syntax error will occur.



SELECT construct logic

```
SELECT
 WHEN day = 'Monday' THEN SAY 'Gym this morning'
 WHEN day = 'Tuesday' THEN SAY 'Cycling this morning'
 WHEN day = 'Wednesday' THEN SAY 'Swimming this morning'
 WHEN day = 'Thursday' THEN SAY 'Running this morning'
 WHEN day = 'Friday' THEN SAY 'Gym this morning'
 WHEN day = 'Saturday' THEN SAY 'Swimming this morning'
 OTHERWISE SAY 'Sleep in'
END
```

This example of the SELECT instruction always evaluates the same variable.

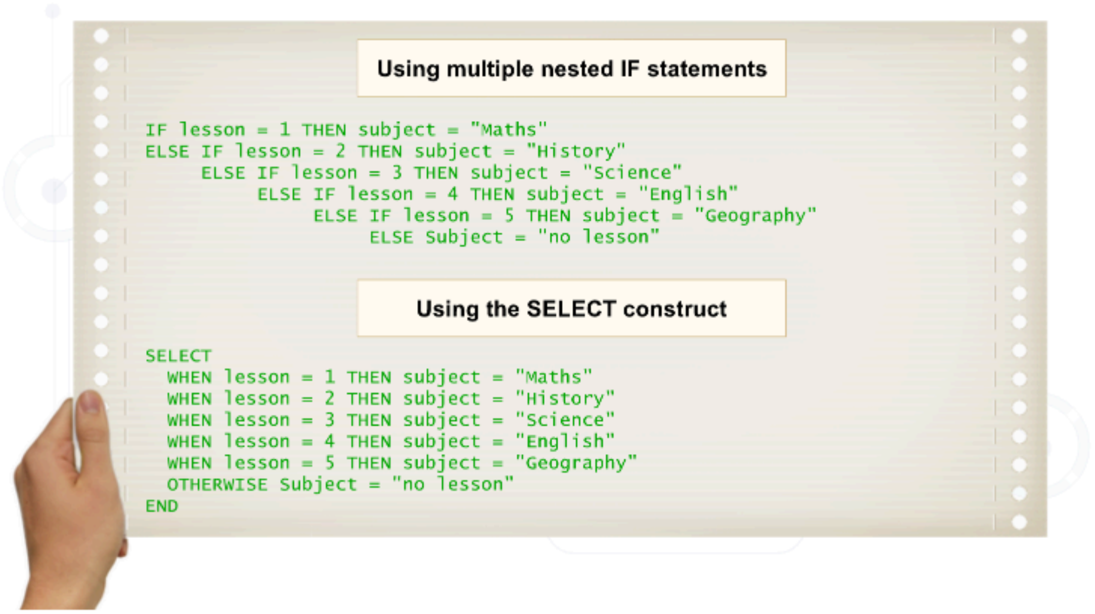


SELECT construct using different variables

```
SELECT
 WHEN engines /= 'OK' THEN SAY 'Engines check failed'
 WHEN flaps /= 'OK' THEN SAY 'Flaps check failed'
 WHEN fuel /= 'OK' THEN SAY 'Fuel check failed'
 WHEN rudder /= 'OK' THEN SAY 'Rudder check failed'
 WHEN runway /= 'OK' THEN SAY 'Runway check failed'
 OTHERWISE SAY 'clear for takeoff'
END
```

This example of the SELECT instruction evaluates different variables.

While multiple WHEN expressions could evaluate as true, only the first one encountered will execute the THEN clause instruction.


Using multiple nested IF statements

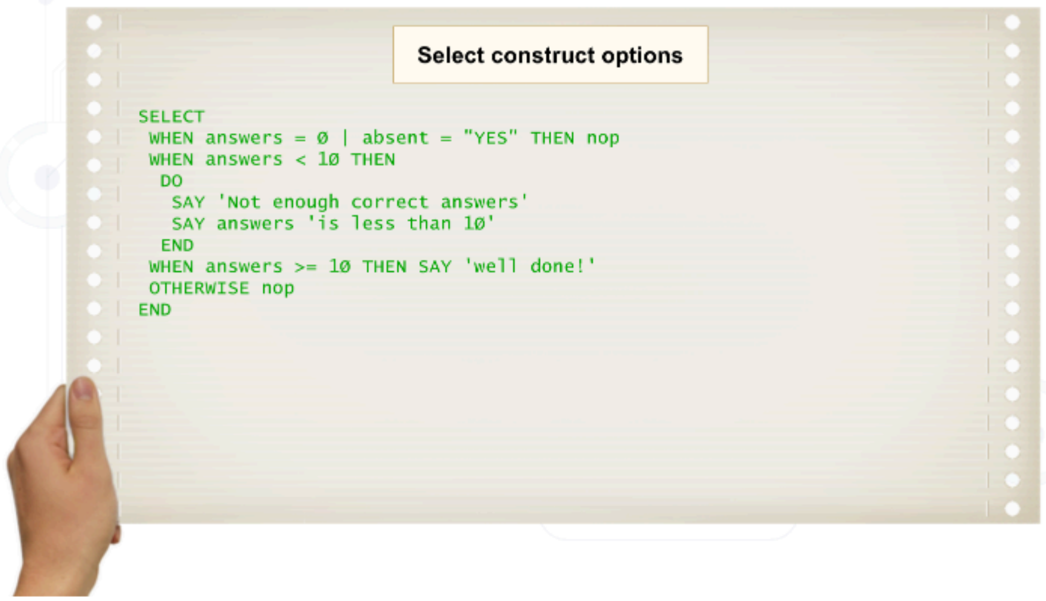
```
IF lesson = 1 THEN subject = "Maths"
ELSE IF lesson = 2 THEN subject = "History"
     ELSE IF lesson = 3 THEN subject = "Science"
          ELSE IF lesson = 4 THEN subject = "English"
               ELSE IF lesson = 5 THEN subject = "Geography"
                    ELSE Subject = "no lesson"
```

Using the SELECT construct

```
SELECT
  WHEN lesson = 1 THEN subject = "Maths"
  WHEN lesson = 2 THEN subject = "History"
  WHEN lesson = 3 THEN subject = "Science"
  WHEN lesson = 4 THEN subject = "English"
  WHEN lesson = 5 THEN subject = "Geography"
  OTHERWISE Subject = "no lesson"
END
```

This example compares multiple IF statements with the SELECT/WHEN/OTHERWISE construct to perform the same logic.

The SELECT statement makes code easier to read, maintain, and debug.

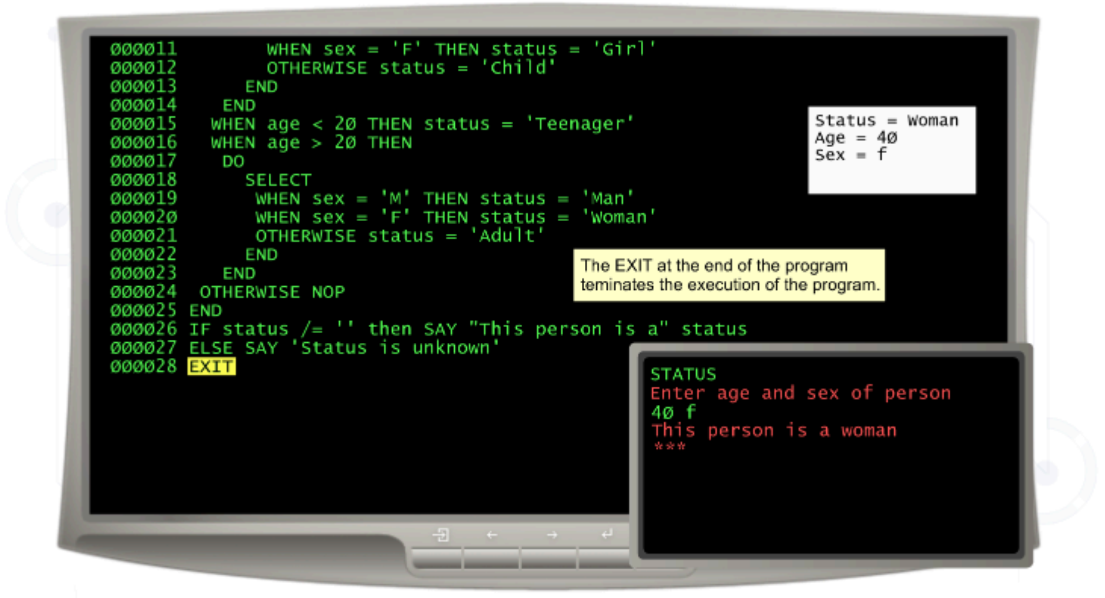


```
SELECT
 WHEN answers = 0 | absent = "YES" THEN nop
 WHEN answers < 10 THEN
  DO
   SAY 'Not enough correct answers'
   SAY answers 'is less than 10'
  END
 WHEN answers >= 10 THEN SAY 'well done!'
 OTHERWISE nop
END
```

Like the IF instruction, SELECT supports the use of the DO and END keywords, NOP, and nesting.

Although the OTHERWISE clause is not always required, it is good practice to code it with the NOP instruction to avoid possible program errors.

WHEN clauses can also evaluate multiple conditions by using Boolean logic operands. This example shows some of these options.



```
       WHEN sex = 'F' THEN status = 'Girl'
       OTHERWISE status = 'Child'
     END
   END
  WHEN age < 20 THEN status = 'Teenager'
  WHEN age > 20 THEN
   DO
     SELECT
      WHEN sex = 'M' THEN status = 'Man'
      WHEN sex = 'F' THEN status = 'Woman'
      OTHERWISE status = 'Adult'
     END
   END
 OTHERWISE NOP
END
IF status /= '' then SAY "This person is a" status
ELSE SAY 'Status is unknown'
EXIT
```

```
Status = Woman
Age = 40
Sex = f
```

The EXIT at the end of the program teminates the execution of the program.

```
STATUS
Enter age and sex of person
40 f
This person is a woman
***
```

This example shows how the SELECT construct works.

Click **Play** to see the program run.