ESCI 386 – IDL Programming for Advanced Earth Science Applications
Lesson 6 – Program Control

**Reading:**  Bowman, Chapters 16

**CODE BLOCKS**
- A code block consists of several lines of code contained between a `BEGIN` and an `END` statement.
- Code blocks can be used with **IF, ELSE, FOR, CASE, and WHILE statements.**
- The `END` statement takes the form of whatever statement began the block
  - ENDIF, ENDELSE, ENDFOR, ENDCASE, or ENDWHILE, depending on the type of block.
  - Writing just `END` instead of `ENDIF`, `ENDFOR`, etc. is legal, but is highly discouraged.

**IF...THEN...ELSE STATEMENTS**
- if...then...else statements have either a single-line form or a block form.
- The single-line form looks like the following examples:
  ```idl```
  ```
  if condition then expression
  if condition then expression1 else expression2
  ```
  ```
- The block form looks like the following examples:
  ```idl```
  ```
  Example #1
  if condition then begin
    expression
  endif
  ```
  ```
  Example #2
  if condition then begin
    expression1
  endif else begin
    expression2
  endelse
  ```
  ```
Example #3

if condition1 then begin
  expression1
endif else if condition2 then begin
  expression2
endif else if condition3 then begin
  expression3
endif else begin
  expression4
endelse

TERNARY OPERATOR

- IDL has a short-hand way of writing simple, IF-THEN-ELSE statement, using the ternary operator, “?:”.
- Then syntax for this is:
  expression with condition ? expression1 : expression2
- If condition is true, then expression1 is substituted for condition before the complete expression on the left-hand-side of the ? character is executed.
- If condition is false, then expression2 is substituted for condition before the complete expression on the left-hand-side of the ? character is executed.
- Example 1:
  \[
  z = (a \text{ gt } b) \ ? a : b
  \]
  will assign the value of \( a \) to \( z \) if \( a \) is greater than \( b \). If not, then \( b \) will be assigned to \( z \).
  o If this were written using IF-THEN-ELSE format it would be
    if (a gt b) then z = a else z = b
- Example 2:
  print, (a \text{ gt } b) ? a : b
  will print the value of \( a \) if \( a \) is greater than \( b \). If not, then \( b \) will be printed.
  o If this were written using IF-THEN-ELSE format it would be
    if (a gt b) then print, a else print, b
TRUE AND FALSE VALUES

- A reminder that IDL doesn’t have specific logical variable for ‘true’ and ‘false’. Instead, it uses the following rules:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte, integer, and long</td>
<td>Odd integers</td>
<td>Zero or even integers</td>
</tr>
<tr>
<td>Floating point and complex</td>
<td>Non-zero values</td>
<td>Zero</td>
</tr>
<tr>
<td>String</td>
<td>Any string with non-zero length</td>
<td>Null string (&quot; &quot;)</td>
</tr>
</tbody>
</table>

- This means that you can use variables as the condition in an IF statement as shown by the examples:

```
  a = 3
  b = 6
  if a then print, ‘hi’ => ‘hi’
  o this evaluates as ‘true’ and prints because a is an odd integer.
  a = 3
  b = 6
  if b then print, ‘hi’ =>
  o this evaluates as ‘false’ and doesn’t print because b is an even integer.
  a = -3.24
  b = 0.0
  if a then print, ‘hi’ => ‘hi’
  o this evaluates as ‘true’ and prints because a is a non-zero floating-point number.
  a = -3.24
  b = 0.0
  if b then print, ‘hi’ =>
  o this evaluates as ‘false’ and doesn’t print because b is floating-point zero.
  a = ‘idl’
  b = ‘’
  if a then print, ‘hi’ => ‘hi’
  o this evaluates as ‘true’ and prints because a is a non-null string.
  a = -3.24
```
\[ b = 0.0 \]

```
if b then print, 'hi' =>
```

- this evaluates as ‘false’ and doesn’t print because \( b \) is a null string.

**CASE STATEMENT**

- Trying to code a multiple if-then-else statement can get cumbersome. IDL has a **CASE** statement which is much easier for this type of construct. It looks like:

```
case grade of
  "A"  : QPA = 4.0
  "B"  : QPA = 3.0
  "C"  : QPA = 2.0
  "D"  : QPA = 1.0
else : QPA = 0.0
endcase
```

- The **CASE** statement will test each case until it gets to one that is true, and will then execute that statement, ignoring all cases below it.

**SWITCH STATEMENT**

- The **SWITCH** statement is similar to the **CASE** statement, except that it tests each case until it gets to one that is true, and then executes that statement, AS WELL AS any subsequent statement regardless of whether or not they are true.

- The example for this, taken from the IDL online documentation, is

```
SWITCH x OF
  1: PRINT, 'one'
  2: PRINT, 'two'
  3: PRINT, 'three'
  4: PRINT, 'four'
ENDSWITCH
```

- which if \( x = 2 \) would result in the printing of “two”, “three”, and “four”.

4
• If the order in the SWITCH were reversed, such as

```
SWITCH x OF
  4: PRINT, 'four'
  3: PRINT, 'three'
  2: PRINT, 'two'
  1: PRINT, 'one'
ENDSWITCH
```

the result would be the printing of “two” and “one” for x = 2.

FOR LOOPS

• IDL has several different constructs for loops.

• The block-form of a FOR loop in IDL takes the form of

```
for i = m, to n, optional_increment do begin
  ...code within loop
endfor
```

Examples:

```
for i = 0, 10 do begin
  d = 2*i
  print, d
endfor
```

would print 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20.

```
for i = 0, 10, 2 do begin
  print, i
endfor
```

would print 0, 2, 4, 6, 8, and 10.

• NOTE: If the code within the loop is only a single line, you can actually write it without using the block form (no begin and end statements). You could instead put everything on a single line, such as

```
for i = 0, 10, 2 do print, i
```
• To increment backward through a loop, use a negative increment interval.

   Example
   for i = 10, 0, -2 do begin
       print, i
   endfor
   would print 10, 8, 6, 4, 2, 0.

WHILE LOOPS
• WHILE loops in IDL take the form of

   while (...some condition...) do begin
       ...code within loop...
   endwhile

   Example:
   i = 0
   while (i le 5) do begin
       print, i
       i = i + 1
   endwhile
   would print 0, 1, 2, 3, 4, and 5.

REPEAT...UNTIL LOOPS
• REPEAT...UNTIL loops in IDL take the form of

   repeat begin
       ...code within loop...
   endrep until (...some condition...)
Example:

```plaintext
i = 0
repeat begin
    print, i
    i = i + 1
end rep until (i ge 5)
would print 0, 1, 2, 3, and 4.
```

- FOR and WHILE loops evaluate the condition or counter at the beginning of the loop. REPEAT...UNTIL loops evaluate the condition at the end of the loop.

BREAKING OUT OF A LOOP OR IF BLOCK

- The **BREAK** procedure will terminate a loop or if-block and carry-on with the rest of the program.
- The **EXIT** procedure will terminate the program.