ARITHMETIC OPERATORS

- The assignment operator in IDL is the equals sign, =.
- IDL uses all the familiar arithmetic operators (+, −, *, /) in the same manner that they are used in other programming languages.
- The exponent or power operator in IDL is the ^ symbol.
  Example: \( a^3 \) is represented as \( a^3 \) in IDL.

INCREMENT AND DECREMENT OPERATORS

- Latter versions of IDL (6 and higher) include the increment and decrement operators (++ and −−).
- The ++ operator adds 1 to it’s operand.
  Example
  \[
  a = 3 \\
  a++ \\
  \text{print, } a => 4
  \]
- The −− operator subtracts 1 to it’s operand.
  Example
  \[
  a = 3 \\
  a-- \\
  \text{print, } a => 2
  \]
- The increment and decrement operators ALWAYS change the value of the operand, regardless of where they occur, although it may change after it performs other operations.
  For example
  \[
  a = 3 \\
  \text{print, } a++ => 3 \text{ ;first prints value of } a, \text{ then increments} \\
  \text{print, } a => 4
  \]
- Only one increment operator can be used at a time per variable.
  - Can’t do \((a++)++\)
COMPOUND OPERATORS

• IDL also has compound operator of the form \( [\text{operator}]= \)

• A statement such as
  \[ b = b \ [\text{operator}] \text{ expression} \]
  can be written equivalently as
  \[ b \ [\text{operator}]= \text{ expression} \]

• Examples of this are:
  • instead of writing
    \[ a = a + b \]
    you can write
    \[ a += b \]
  • instead of writing
    \[ a = a - b \]
    you can write
    \[ a -= b \]
  • instead of writing
    \[ a = a*b \]
    you can write
    \[ a *= b \]
  • instead of writing
    \[ a = a/b \]
    you can write
    \[ a /= b \]
  • instead of writing
    \[ a = a^2 \]
    you can write
    \[ a ^= 2 \]

• Compound operators are just a short-hand way to write longer expressions. You don’t have to use them, as you can always write out the longer version of the expressions.
MODULO OPERATOR

- To find the modulus (remainder) of two numbers, use the MOD operator,
  
  print, 5 mod 3 => 2
  print, 6 mod 3 => 0
  print, 3.45 mod 2 => 1.45

- The modulo operator is handy for determining if one number is evenly divisible by another. If \( a \mod b \) is equal to zero, then either \( a \) is zero, or \( a \) is evenly divisible by \( b \).

TRUE AND FALSE VALUES

- 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 (‘’)</td>
</tr>
</tbody>
</table>

RELATIONAL OPERATORS

- In IDL the relational operators are:
  
  eq  equal to
  ne  not equal to
  gt  greater than
  ge  greater than or equal to
  lt  less than
  le  less than or equal to
• The relational operators return a BYTE value of 0 if they evaluate as ‘false’ and a BYTE value of 1 if they evaluate as ‘true’.

Examples:
print, 2 eq 3 => 0
print, 2 ne 3 => 1
print, 3 lt 2 => 0
print, 3 ge 2 => 1
print, 2 ge 2 => 1

BOOLEAN (LOGICAL) OPERATORS
• IDL also has Boolean (logical) operators for “and,” “or,” and “not.”
  o These are
    & & is Boolean ‘and’
    || is Boolean ‘or’
    ~ is Boolean ‘not’

Examples:
a = 2
b = 3
print, (a eq 4) || (b eq 3) => 1
print, (a eq 3) || (b eq 4) => 0
print, (a eq 2) && (b eq 3) => 1
print, ~(a eq 2) => 0

• IDL also has operators AND, OR, and NOT that are bit-wise operators as opposed to logical operators. The difference between the logical operators and bit-wise operators is esoteric for our purposes. You can use either, though I recommend using the &, ||, and ~ forms.

PRECEDENCE OF OPERATORS
• Operators are executed in order of priority, from highest to lowest.
• If two operators have the same priority, then they are executed from left to right.
• The table below is copied from Table 12-9 of the IDL online help.
### Priority | Operator |
|----------|----------|
| **First** (highest) | ( ) (parentheses, to group expressions)  
[ ] (brackets, to concatenate arrays) |
| **Second** | . (structure field dereference)  
[ ] (brackets, to subscript an array)  
( ) (parentheses, used in a function call) |
| **Third** | * (pointer dereference)  
^ (exponentiation)  
++ (increment)  
-- (decrement) |
| **Fourth** | * (multiplication)  
# and ## (matrix multiplication)  
/ (division)  
MOD (modulus) |
| **Fifth** | + (addition)  
- (subtraction and negation)  
< (minimum)  
> (maximum)  
NOT (bitwise negation)  
~ (logical negation) |
| **Sixth** | EQ (equality)  
NE (not equal)  
LE (less than or equal)  
LT (less than)  
GE (greater than or equal)  
GT (greater than) |
| **Seventh** | AND (bitwise AND)  
OR (bitwise OR)  
XOR (bitwise exclusive OR) |
| **Eighth** | && (logical AND)  
|| (logical OR) |
| **Ninth** | ?: (conditional expression) |

**Taken from Table 12-9 of IDL Online Help**

### ADDITIONAL CONSIDERATIONS

- **DON’T GO OVERBOARD ON PARENTHESES!**
  - If you understand the precedence of operators, there is no need to make your expressions difficult to read by using a bunch of parentheses.

  **Example:**

  Write

  \[ y = 2.0 \times x^{2.5} \times \exp \left( -a \times b / x \right) \]

  **instead of**

  \[ y = (2.0 \times (x^{2.5})) \times \exp \left( -a \times b / x \right) \]
Expressions such as
\[ y = \frac{x}{a \times b} \]
can be written without parentheses as
\[ y = \frac{x}{a \div b} \]
The second expression actually executes faster on the computer, too!

- **Whenever you are doing division, make sure at least one expression is a floating-point number.** This will help you avoid the dreaded “integer division” problem.

  Write
  \[ y = \frac{x \times y}{2.0} ; \quad \text{This is safe.} \]
  instead of
  \[ y = \frac{x \times y}{2} ; \quad \text{This is dangerous. May have integer division problem.} \]

- **DON’T USE FLOATING POINTS FOR EXPONENTS unless you have to.**

  Write
  \[ y = x^2 \]
  instead of
  \[ y = x^2.0 \]