REXX Terms, Variables, and Operators

By proceeding with this courseware you agree with these terms and conditions. Interskill Learning Pty. Ltd. © 2019

Objectives

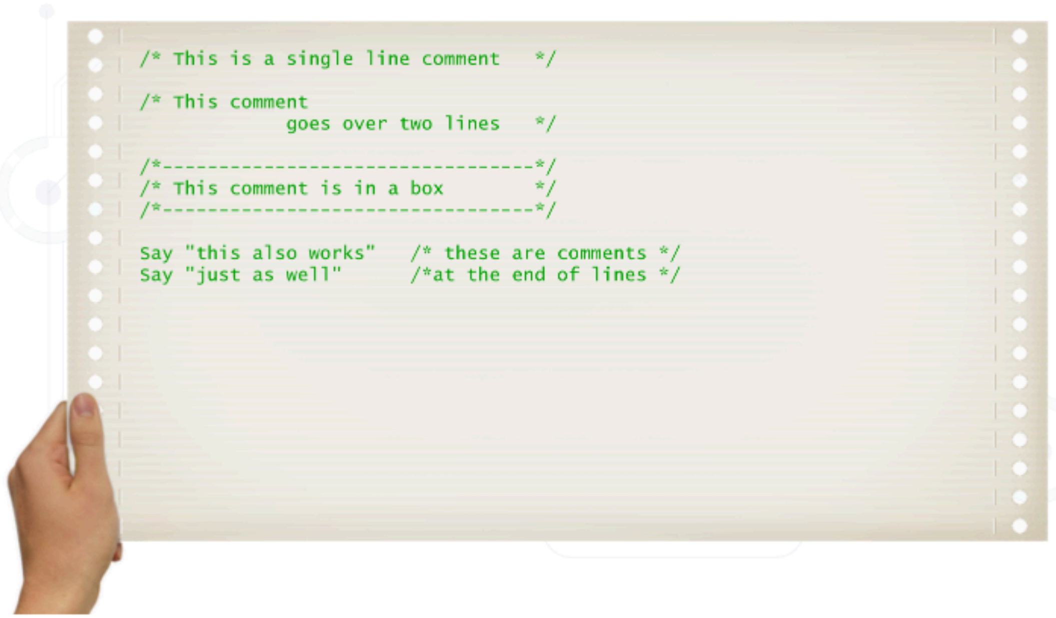
REXX Terms, Variables, and Operators

In this module, you will be introduced to the common terms and variables that are used in REXX.

You will also look at the operators that can be coded for arithmetic, comparison, logical, and string operations in a REXX program.

After completing this module, you will be able to:

- Recognize REXX Terms and Variables
- Identify REXX Operators

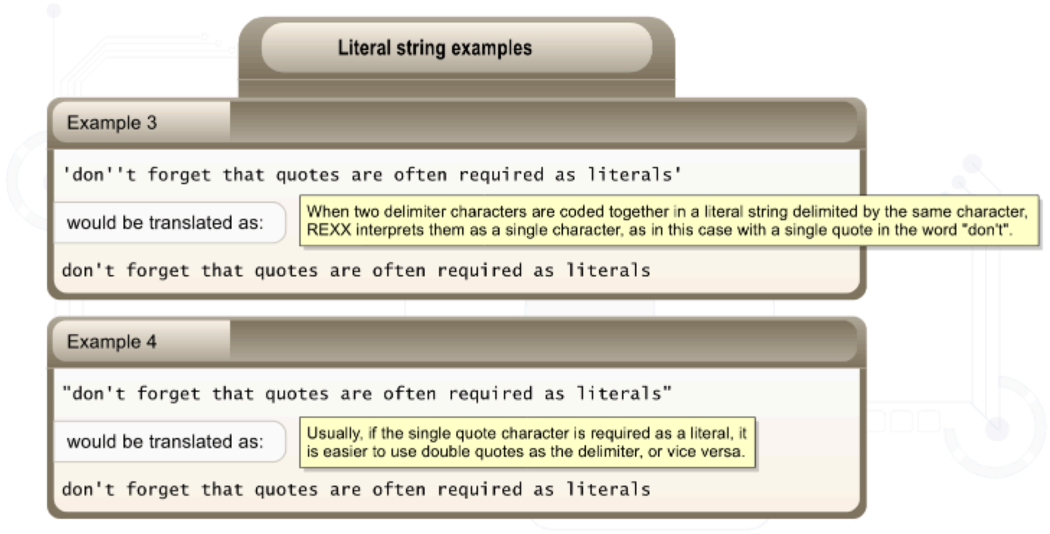


```
/* This is a single line comment   */

/* This comment
             goes over two lines   */

/*---------------------------------*/
/* This comment is in a box        */
/*---------------------------------*/

Say "this also works"   /* these are comments */
Say "just as well"      /*at the end of lines */
```

A REXX comment is a sequence of characters delimited by `/*` and `*/`.

Comments can, in turn, have other comments nested within them. They can be placed anywhere in a REXX program and can continue over multiple lines.

The REXX interpreter ignores comments.



Literal string examples

Example 3

```
'don''t forget that quotes are often required as literals'
```

would be translated as:

When two delimiter characters are coded together in a literal string delimited by the same character, REXX interprets them as a single character, as in this case with a single quote in the word "don't".

```
don't forget that quotes are often required as literals
```

Example 4

```
"don't forget that quotes are often required as literals"
```

would be translated as:

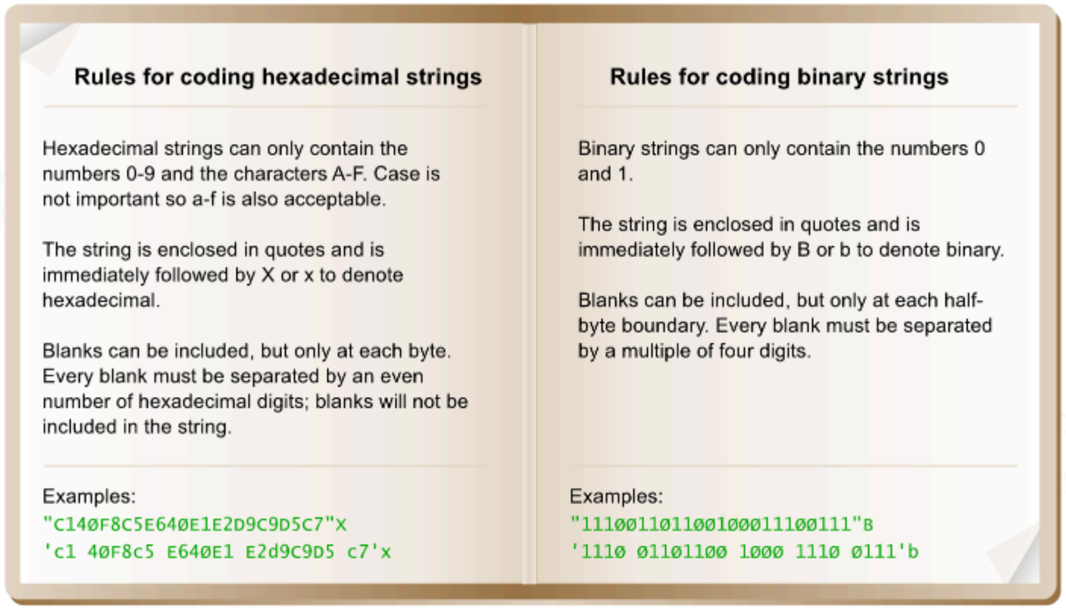
Usually, if the single quote character is required as a literal, it is easier to use double quotes as the delimiter, or vice versa.

```
don't forget that quotes are often required as literals
```

When a REXX program is executed, the interpreter "sees" everything as a symbol or variable. Any symbol or variable that has not been assigned a value is assigned a default value of the variable name with all alphabetic characters converted to uppercase.

If letters or character strings are to be treated as literals rather than symbols, they must be identified as literals by enclosing the relevant character string in single (') or double (") quotation marks. When the interpreter encounters one of these characters, it reads all the characters up to, but not including, the next quotation mark of the same type and interprets them as literals, exactly as typed.

Click **Play** to see some examples of literal strings.



Rules for coding hexadecimal strings

Hexadecimal strings can only contain the numbers 0-9 and the characters A-F. Case is not important so a-f is also acceptable.

The string is enclosed in quotes and is immediately followed by X or x to denote hexadecimal.

Blanks can be included, but only at each byte. Every blank must be separated by an even number of hexadecimal digits; blanks will not be included in the string.

Examples:

```
"C140F8C5E640E1E2D9C9D5C7"X
'c1 40F8c5 E640E1 E2d9C9D5 c7'x
```

Rules for coding binary strings

Binary strings can only contain the numbers 0 and 1.

The string is enclosed in quotes and is immediately followed by B or b to denote binary.

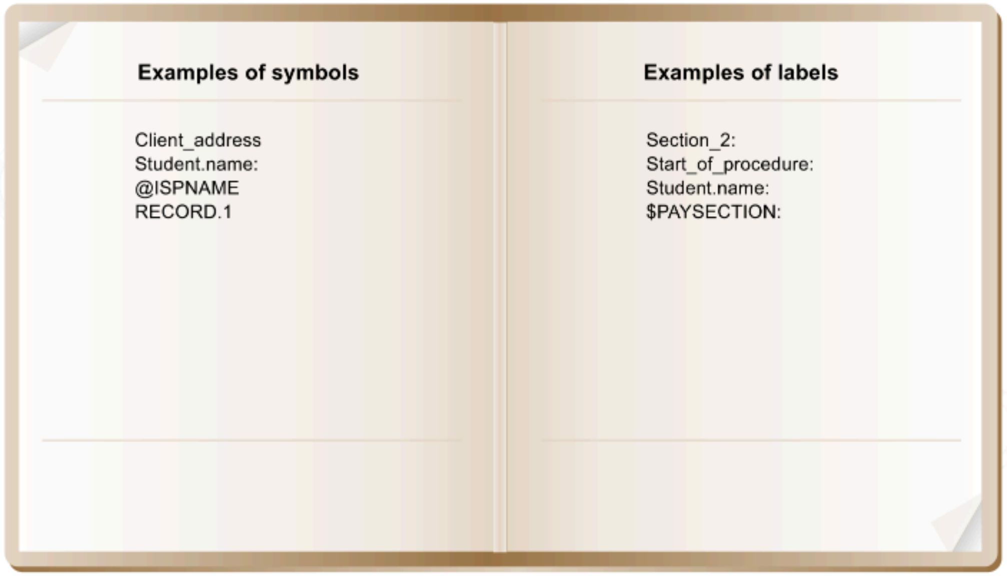
Blanks can be included, but only at each half-byte boundary. Every blank must be separated by a multiple of four digits.

Examples:

```
"111001101100100011100111"B
'1110 01101100 1000 1110 0111'b
```

Hexadecimal and binary strings can also be defined. With some exceptions, these are coded like literal strings.

Listed above are the rules for defining hexadecimal and binary strings.



Examples of symbols

Client_address
Student.name:
@ISPNAME
RECORD.1

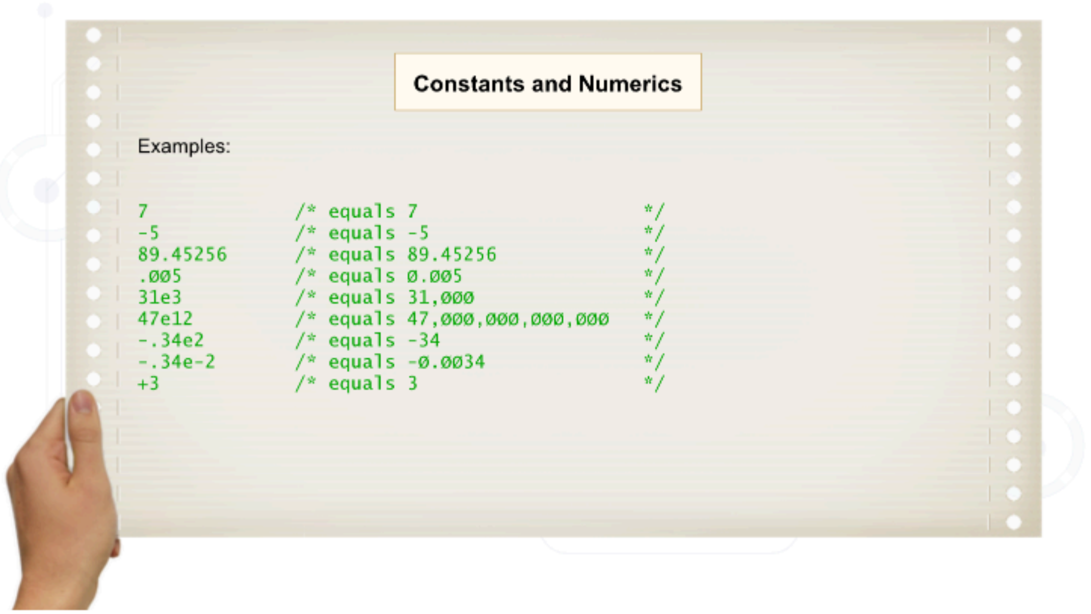
Examples of labels

Section_2:
Start_of_procedure:
Student.name:
$PAYSECTION:

A REXX symbol consists of a group of characters. The characters can be any valid alphanumeric or any `@`, `#`, `$`, `¢`, `!`, `?`, `.` or `_` characters.

A symbol must begin with an alpha character or any of the special characters, except the period. Symbols can be variables, labels, constants, or REXX keywords.

A label is a symbol with a colon (`:`) on the end.



Constants and Numerics

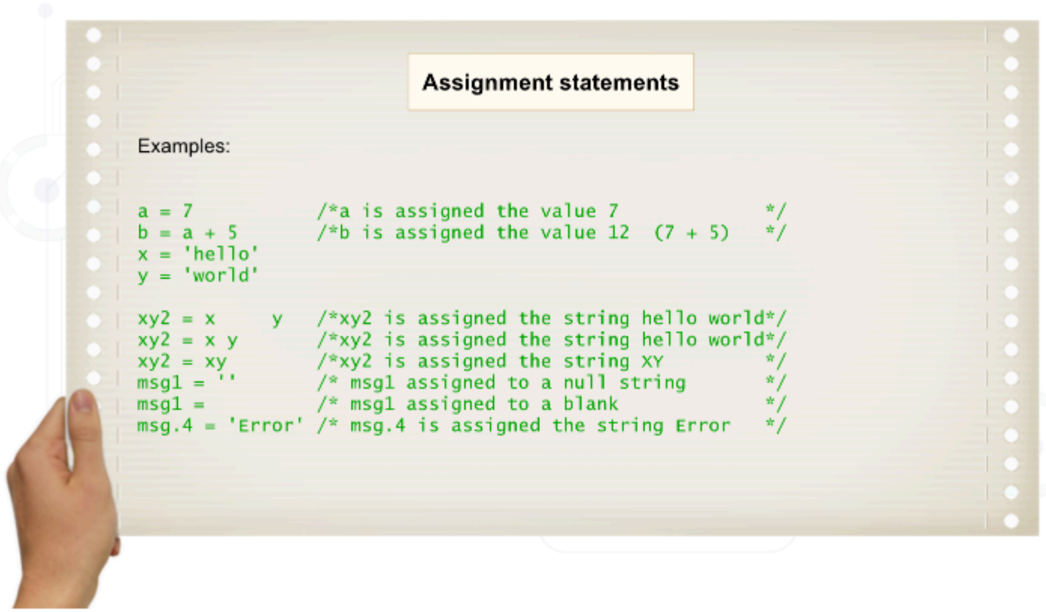
Examples:

```
7             /* equals 7                         */
-5            /* equals -5                        */
89.45256      /* equals 89.45256                  */
.005          /* equals 0.005                     */
31e3          /* equals 31,000                    */
47e12         /* equals 47,000,000,000,000        */
-.34e2        /* equals -34                       */
-.34e-2       /* equals -0.0034                   */
+3            /* equals 3                         */
```

A constant is a symbol that starts with a numeric character or a period. Constants are also referred to as numerics.

Numerics can be expressed as whole numbers, decimals, and exponents.

Listed above are several examples of numerics and the value that the REXX interpreter will assign them.



Assignment statements

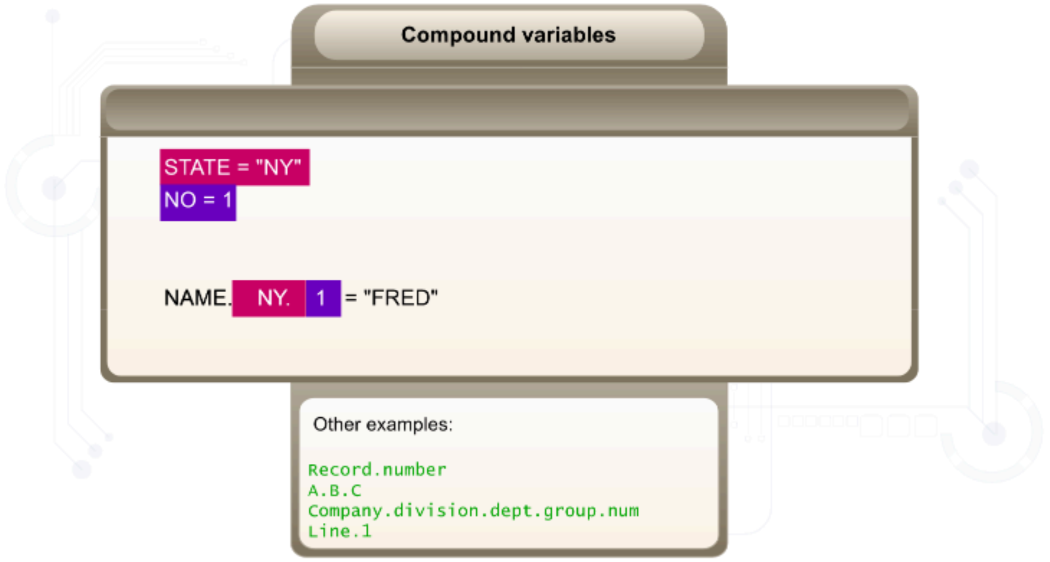
Examples:

```
a = 7          /*a is assigned the value 7          */
b = a + 5      /*b is assigned the value 12  (7 + 5)   */
x = 'hello'
y = 'world'

xy2 = x     y   /*xy2 is assigned the string hello world*/
xy2 = x y       /*xy2 is assigned the string hello world*/
xy2 = xy        /*xy2 is assigned the string XY        */
msg1 = ''       /* msg1 assigned to a null string      */
msg1 =          /* msg1 assigned to a blank            */
msg.4 = 'Error' /* msg.4 is assigned the string Error   */
```

A REXX assignment statement must contain a valid REXX symbol as the target or variable name of the assignment. The right side of an assignment can contain another variable, constant, compound variable, or expression composed of a combination of these.

When assigning a variable to multiple values, unless they are defined as literals, REXX will regard multiple spaces as a single space.

Shown here are some examples of assignment statements, and the use of abuttal characters and methods.

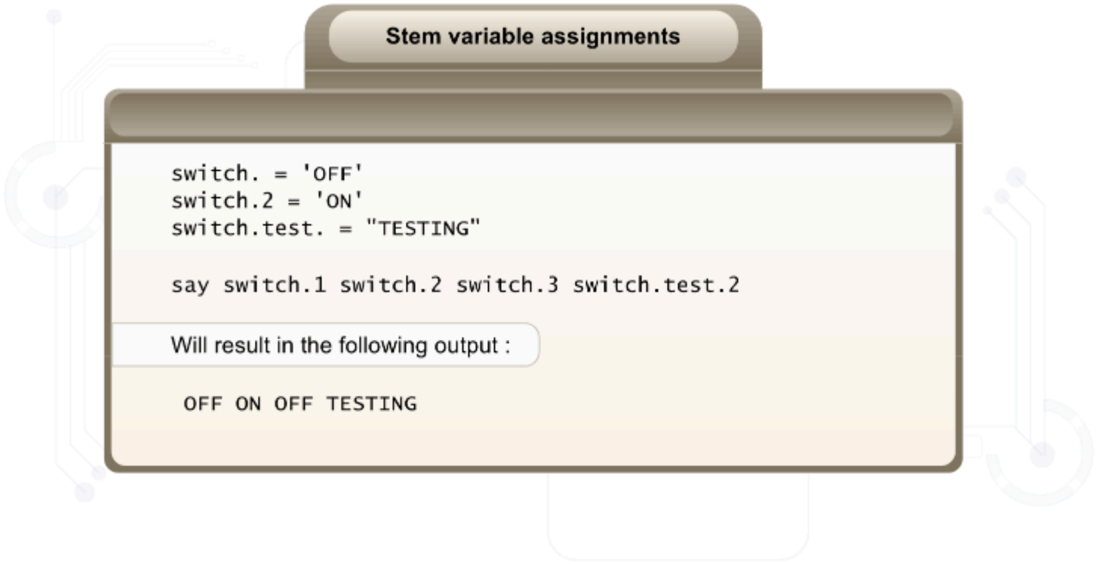


```
Record.number
A.B.C
Company.division.dept.group.num
Line.1
```

A compound variable is the REXX equivalent of an array. It consists of a stem value with a period as the last character, which is followed by at least one other character called the tail. When the tail begins with a numeric, it is often referred to as the index. As with REXX variables, compound variables must not begin with a numeric character.

Compound variables can contain multiple stems and tails. A unique feature of the compound variable is that all qualifiers, except the primary stem, are treated as variables and will have their values substituted if assigned. The qualifiers are the characters between each period. A compound variable can be a maximum of 250 characters in length before and after variable substitution.

Click **Play** to see some examples of compound variables.



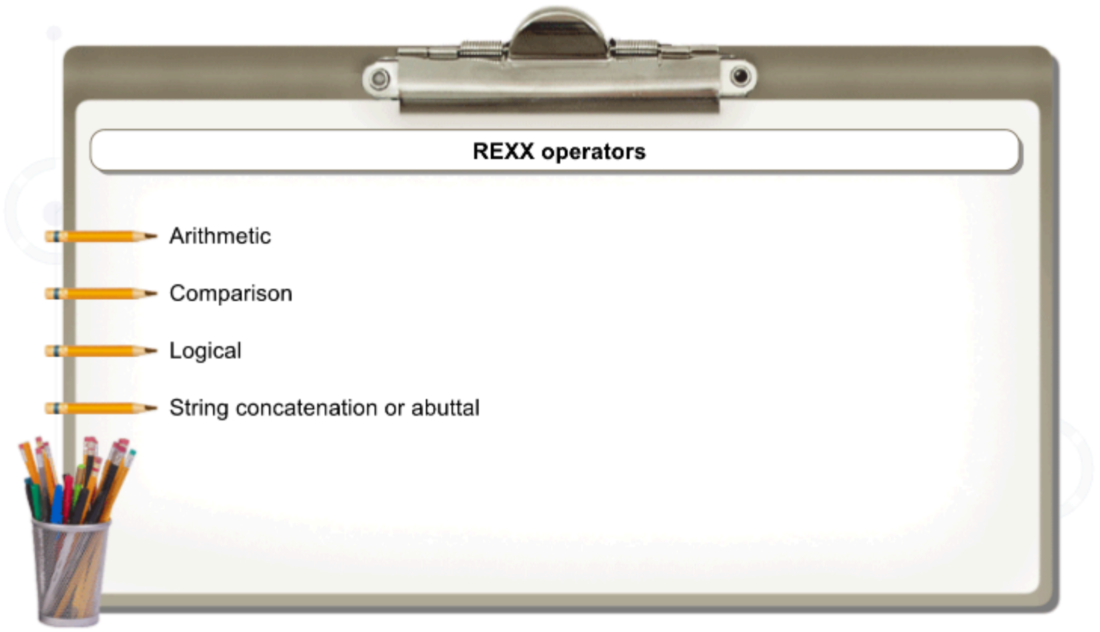
```
switch. = 'OFF'
switch.2 = 'ON'
switch.test. = "TESTING"

say switch.1 switch.2 switch.3 switch.test.2
```

Will result in the following output :

```
 OFF ON OFF TESTING
```

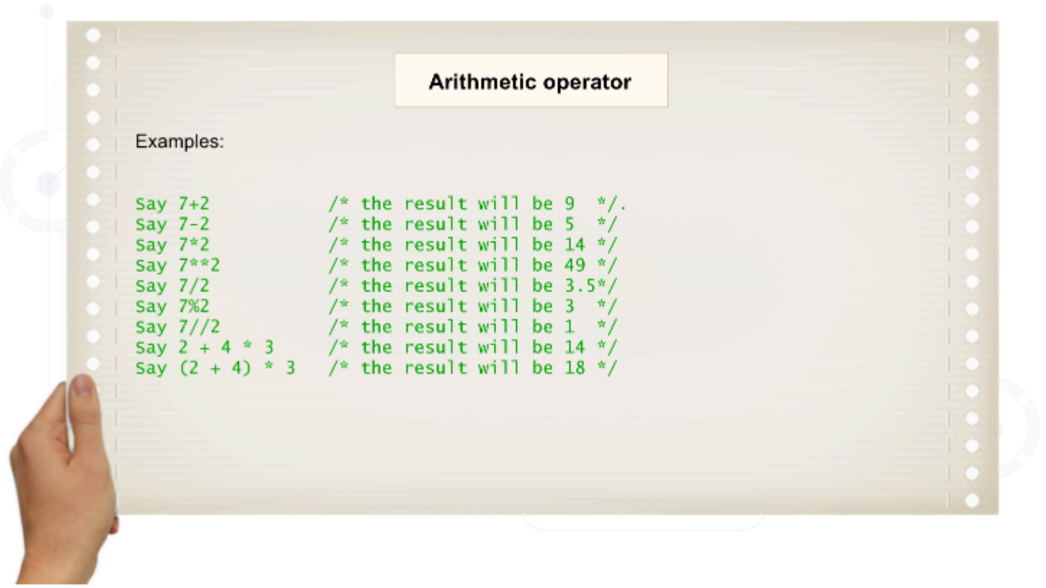
A stem also has a useful property and this is that it always ends in a period.

When a stem variable is used as the target of an assignment, all compound variables that do or could start with that stem are set to the assigned value.


REXX operators

- Arithmetic
- Comparison
- Logical
- String concatenation or abuttal

An operator is a symbol that defines an operation. REXX operators are divided into the groups listed here.



Arithmetic operator

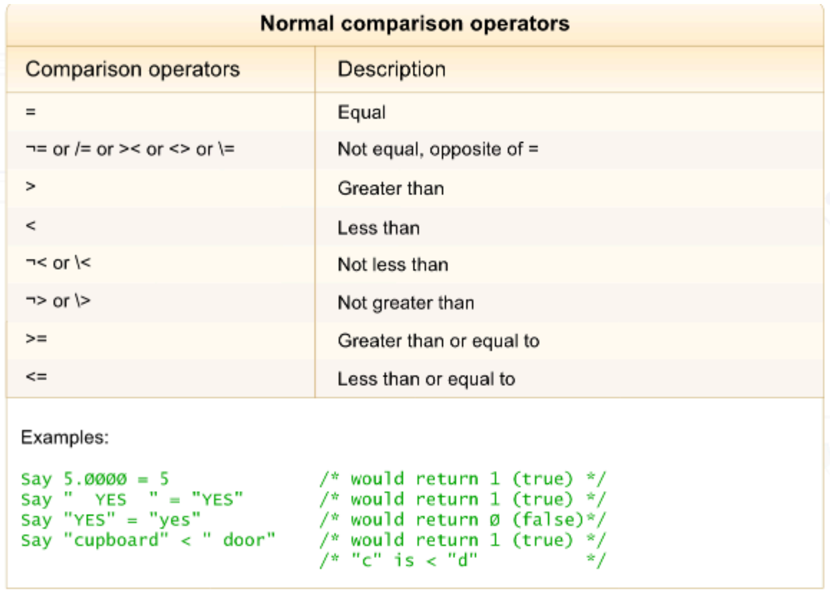
Examples:

```
Say 7+2          /* the result will be 9  */.
Say 7-2          /* the result will be 5  */
Say 7*2          /* the result will be 14 */
Say 7**2         /* the result will be 49 */
Say 7/2          /* the result will be 3.5*/
Say 7%2          /* the result will be 3  */
Say 7//2         /* the result will be 1  */
Say 2 + 4 * 3    /* the result will be 14 */
Say (2 + 4) * 3  /* the result will be 18 */
```

Arithmetic operators perform arithmetic functions. By default, numeric operations are performed to nine significant places. The REXX interpreter can use the standard operators of `+`, `-`, `*`, and `/`, as well as the following special operators:

`%` indicates the integer component of a division operation
`//` indicates the remainder of a division operation
`**` indicates exponentiation or "to the power of"; the power value must be a whole number

REXX arithmetic follows the standard mathematical precedence of `**`, `*`, `//`, `%`, `/`, `+`, `-`, and left to right, but parentheses (`()`) can be used to alter the order of precedence. Spaces coded



Normal comparison operators

Comparison operators	Description
=	Equal
¬= or /= or >< or <> or \=	Not equal, opposite of =
>	Greater than
<	Less than
¬< or \<	Not less than
¬> or \>	Not greater than
>=	Greater than or equal to
<=	Less than or equal to

Examples:

```
Say 5.0000 = 5            /* would return 1 (true) */
Say "  YES  " = "YES"     /* would return 1 (true) */
Say "YES" = "yes"         /* would return 0 (false)*/
Say "cupboard" < " door"  /* would return 1 (true) */
                          /* "c" is < "d"          */
```

Numerous comparison operators are available in REXX. Most of them are much the same as other languages. REXX also accepts a number of forms for the logical `NOT` operator. Note the following:

- When using REXX on a mainframe, the most common `NOT` operator is the symbol Shift-6 on a 3270 series terminal.
- As most PCs do not have this character, the backslash (`\`) symbol can be used. This character is used in this course.
- When the expression is unambiguous, the forward slash (`/`) character can also be used.
- When comparing numerics, the dual characters of `<>` or `><` can be used.

Strict comparison operators

Comparison operators	Description
==	Strictly equal
¬== or \==	Not strictly equal, opposite of ==
>>	Strictly greater than
<<	Strictly less than
¬<< or >>=	Not strictly less than, or strictly greater than, or equal to
¬>> or <<=	Not strictly greater than, or strictly less than, or equal to

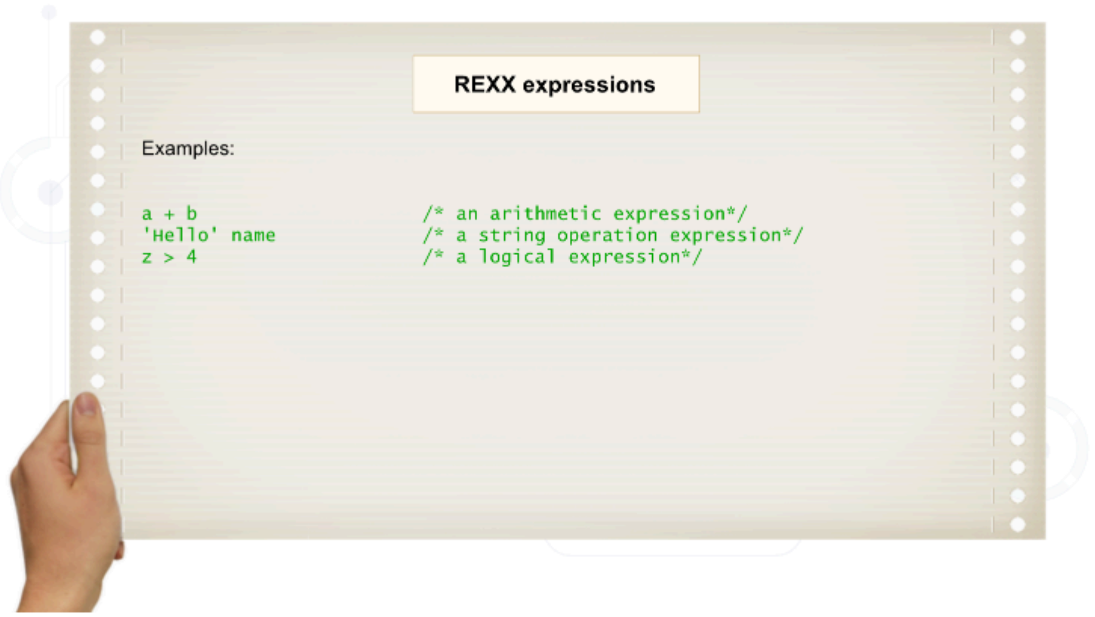
Examples:

```
Say 05.0000 = 5             /* would return 1 (true) */
Say "  YES  " = "YES"       /* would return 1 (true) */
Say "YES" = "yes"           /* would return 0 (false)*/
Say "cupboard" < " door"    /* would return 1 (true) */
                            /* "c" is < "d"          */

Say 05.0000 == 5            /* would return 0 (false)*/
Say "  YES  " == "YES"      /* would return 0 (false)*/
Say "cupboard" << " door"   /* would return 0 (false)*/
                            /* "c" is > " " (space)  */
```

When the strings being compared are both numeric, REXX performs a numeric comparison. Other strings are compared on a character by character basis. String comparisons are case sensitive and the comparison order depends on the character set in use on the platform where the REXX program is running. This may be ASCII or EBCDIC.

Normal comparisons usually ignore leading and trailing blanks in string comparisons, and leading and trailing zeros in numeric comparisons. However, if strict comparison operands are used, that is, double comparison operators, the entire string is compared byte for byte.

Listed above are the strict comparison operators and some examples of normal and strict comparisons.



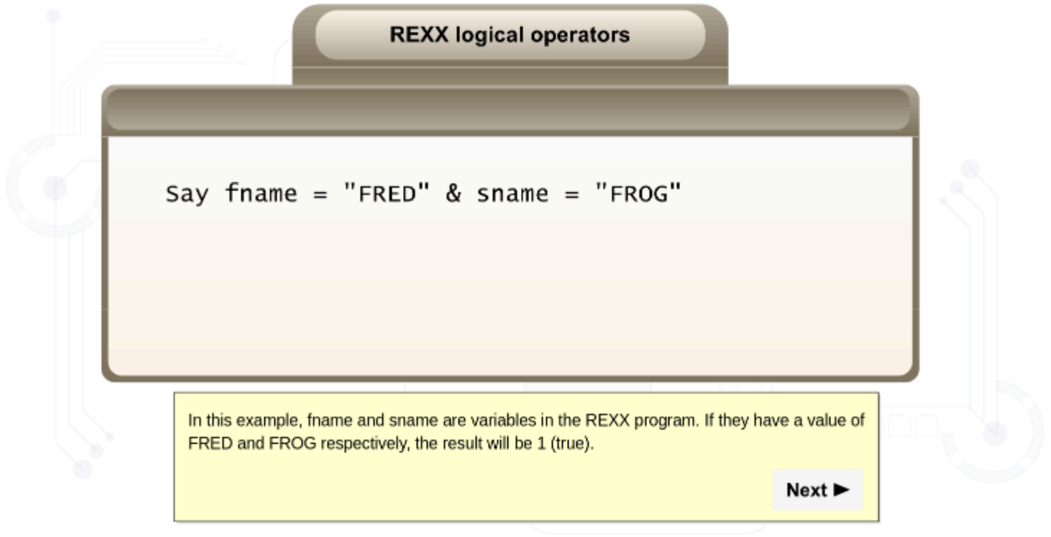
REXX expressions

Examples:

```
a + b                      /* an arithmetic expression*/
'Hello' name               /* a string operation expression*/
z > 4                      /* a logical expression*/
```

A REXX expression consists of a combination of operators, symbols, and parentheses.

Shown here are some examples of REXX expressions.

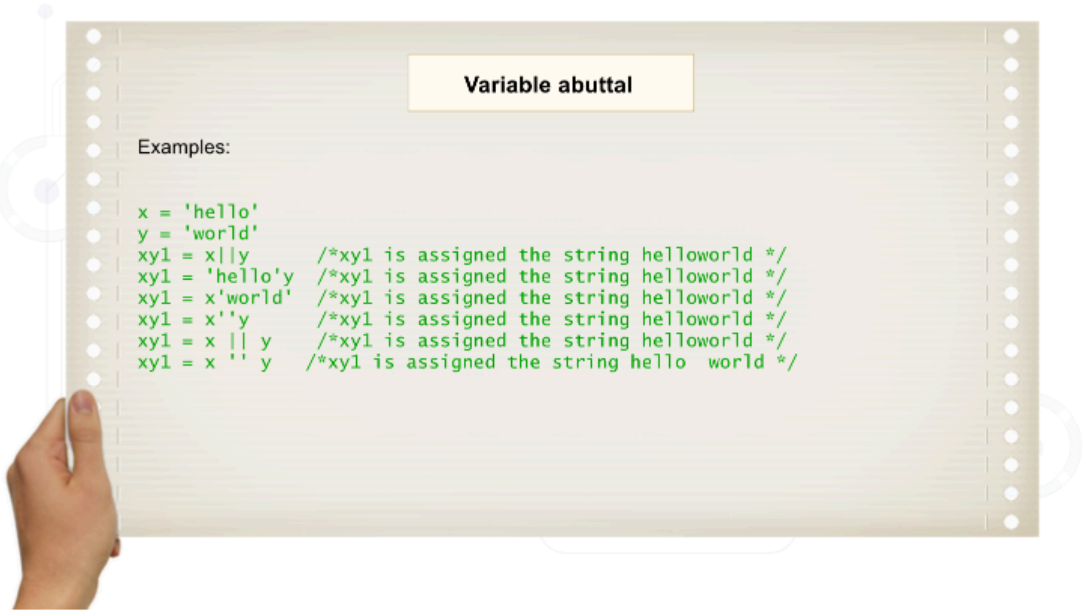


```
Say fname = "FRED" & sname = "FROG"
```

In this example, fname and sname are variables in the REXX program. If they have a value of FRED and FROG respectively, the result will be 1 (true).

Logical operators enable two expressions to be joined by a logical "and", "or", or "exclusive or" . These are generally used in conjunction with logical expressions to determine a multiple condition test for a true/false value. The three logical operators used by REXX are:

& Logical "and" means if both expressions are true, the result is true; otherwise, the result is false. These are processed first, but parentheses can be used to alter the order of precedence.
| Logical "or" means if either or both expressions are true, the result is true; otherwise, the result is false.
&& Logical "exclusive or" means if either, but not both, expressions are true, the result is true; otherwise the result is false.



Variable abuttal

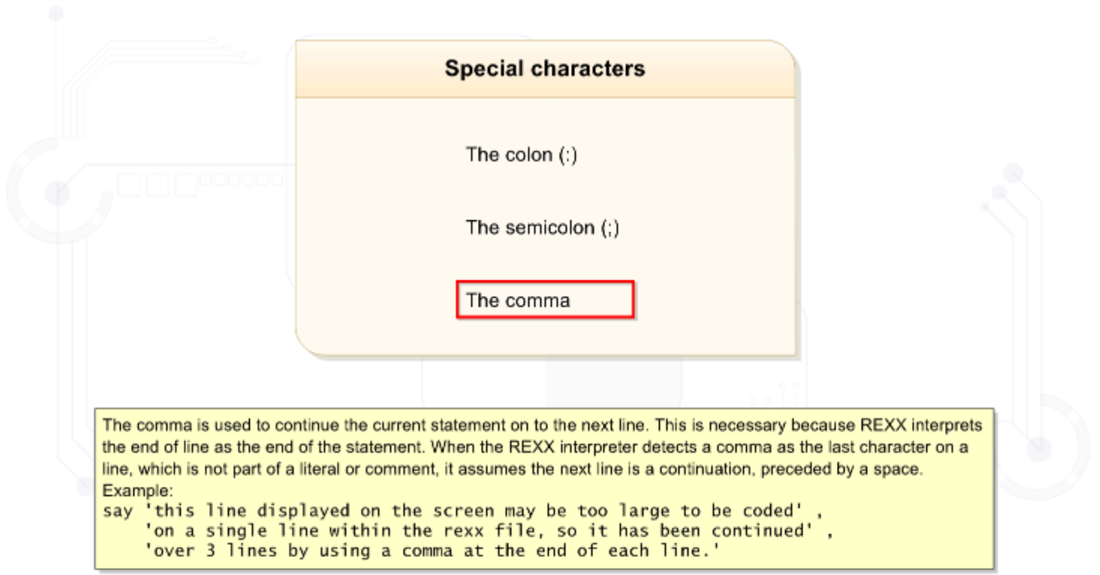
Examples:

```
x = 'hello'
y = 'world'
xy1 = x||y       /*xy1 is assigned the string helloworld */
xy1 = 'hello'y  /*xy1 is assigned the string helloworld */
xy1 = x'world'  /*xy1 is assigned the string helloworld */
xy1 = x''y       /*xy1 is assigned the string helloworld */
xy1 = x || y    /*xy1 is assigned the string helloworld */
xy1 = x '' y   /*xy1 is assigned the string hello  world */
```

When an expression in an assignment statement is required to join two values together without any spaces, double vertical bars (||) can be used as a concatenation (abuttal) characters. Any spaces coded between literals or symbols on either side of abuttal characters, will be removed.

A double ("") string can also be used, but only between symbols that are abutted to them. Alternatively, literals can be abutted to variables by not coding any spaces between them.

Shown here are some examples of abuttal characters.



The comma is used to continue the current statement on to the next line. This is necessary because REXX interprets the end of line as the end of the statement. When the REXX interpreter detects a comma as the last character on a line, which is not part of a literal or comment, it assumes the next line is a continuation, preceded by a space.
Example:

```
say 'this line displayed on the screen may be too large to be coded' ,
    'on a single line within the rexx file, so it has been continued' ,
    'over 3 lines by using a comma at the end of each line.'
```

The colon, semicolon, and comma are special characters that have specific meanings when used in REXX code.

Mouse-over each character for a description.