**REXX Notes**

**sites**

**Tutorials**

**Abends**

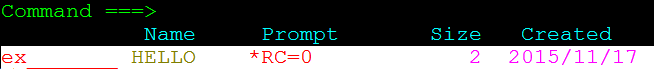
**Utilities –**

**Example Programs**

-----------------------------------------------------------------------------------------------------------------------------

**Imp Points**

* **ABBREVATIONS**
* Supply EX before rexx program member in PDS to run the REXX pgm.



* Comments always start /\* and end \*/.
* Please note that starting letters of all rexx programs must be /\* REXX \*/
* If you want two instructions on one line you will need to put a semi-colon between them.
* Line continuation character is **Comma (,) for easytrive.**

**It is + for IDCAMS.**

**REXX and sort card it is ,**

* The maximum length of a literal is platform specific but it will be at least 100 bytes and normally considerably more.
* **SAY "To print without" || "gaps use these."** This use of Abuttal prints without any spaces
* Simple variable names can contain these chars.

A - Z, a - z, 0 - 9, !, ?, #, $, @ & \_

0 - 9 can not be used for the first character.

All lowercase chars. are converted to uppercase.

Variables are initialized with their name!

**EG:-**

SAY “New\_var=” new\_var

ANS. NEW\_var= new\_var

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------END--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**->SAY** instruction

template: **SAY** expression

EG:

1. **SAY** "print this line"

2. **SAY** "Contents of A=" A

**-> Rules**

There really are very few rules when it comes to coding REXX. However it is important to understand the few that there are.

1. The system presumes that each line is a new instruction unless you state you are continuing on another line by the use of a comma (we will see this in a minute).
2. If you want two instructions on one line you will need to put a semi-colon between them.
3. You can indent your instructions as much as you like; this is very useful when you are coding loops.
4. You can also have as many blank lines as you like. These can be used to section your code and make it far more readable.

**Concatenation**A very important concept it the automatic concatenation that applies within REXX. This is not only used for printing but in lots of other situations. The basic idea is that if you put two things (TOKENS) next to each other (in this example two literal strings), then they will be connected with a space. Note that they are always joined by one space regardless of how many spaces there are between the literal strings.   
**Abuttal**  
If you do not wish to have a space between the two tokens you must code two vertical lines (OR symbol). This system can apply to numbers as well, we haven't looked at these yet but what do you think A would equal here?

**A = 5 || 7**

* **Variables**

1. **Naming Convention-** If your variable name contains lower case characters then these are automatically converted into capitals. See here how the variable TOTAL\_a is used in one line and on the next it is referred to a TOTAL\_A. Doing this for real would be considered a very sloppy habit but it’s quite legal (like a lot of sloppy habits).
2. **Initialized Value**

This is something worth remembering, in REXX you can use a variable even if you have not given it a value. In the example here we have only used NEW\_var on a SAY statement. So what does it print? The answer is its own name (in capitals of course).

1. **Data Type**

Another rather weird thing about REXX is that you do not have to tell it whether your variable is going to contain a string of characters or a number - it just doesn't care!

* **Arithmetic Operators**

/\* REXX program ARITHMETIC operators. \*/

SAY "ADD: 5 + 2 =" 5 + 2 ; SAY "SUBTRACT:2 - 5 =" 2 - 5

SAY "MULTIPLY:5 \* -2 =" 5 \* -2 ; SAY "DIVIDE: 5 / +2 =" 5 / +2

SAY "INT. DIV.5 % 2 =" 5 % 2 ; SAY "REMAINER:5 // 2 =" 5 // 2

SAY "POWER: 5 \*\* 2 =" 5 \*\* 2

SAY "ABUTTAL:5 || 2'00' =" 5 || 2'00'

SAY "USE BRACKETS FOR PRIORITY:" 7 || ((2 + 5) \* 5)/3

----------------------------

ADD: 5 + 2 = 7

SUBTRACT:2 - 5 = -3

MULTIPLY:5 \* -2 = -10

DIVIDE: 5 / +2 = 2.5

INT. DIV.5 % 2 = 2

REMAINER:5 // 2 = 1

POWER: 5 \*\* 2 = 25

ABUTTAL:5|| 2'00' = 5200

USE BRACKETS FOR PRIORITY: 711.6666667

Operators

Here you can see the different mathematical operators available. It all looks pretty straight forward until you come down to ABUTTAL - what is this doing here you may say? Well I said you could use concatenations for more than just pretty SAY statements! Two types of abuttal are used here, one using the '||' symbol and the other is achieved by typing the CONSTANT next to the literal string. We haven't seen this method of abuttal before. Note the last number in the example 711.666667, the precision here means we get a total of 9 digits. This is an environmental option which we can change with the NUMERIC keyword we'll see this later.

Operator Priorities & Brackets

These can get silly, I say that because we could spend a lot of time learning them and then writing very fancy code which works and is very clever. Clever but unreadable when two months later when you can't remember which operator has priority over its neighbour. The solution is simple use brackets even when they may not be needed! If you want to be an anorak here are the priorities, when two are the same, the priority is from left to right.

8] + - (as a prefix) 5] + - (not as a prefix)

7] \*\* 4] Abuttal.

* Expresssions

What is an expression?

Simply its several bits of code which will be used to work out one thing, that 'thing' may be a literal string or a number. Let's take an assignment instruction for example

**OUTLINE = 'The Total is =' L \* 2 + W \* 2 + ExtraBit**

Everything here except the OUTLINE = is one expression. It contains constants, literal strings, arithmetic operators and variables but all the values will be shrunk down to one literal string that in this case will be put into the variable called OUTLINE. We could however print it by coding SAY at the beginning rather than OUTLINE =. Later we will look at functions and the arguments, which are their inputs. Normally we have just a single variable as an argument but in fact it will take a very large and complex expression.

* **The PULL instruction**

/\* REXX program to show the PULL instruction \*/

SAY "Program to multiple 2 numbers"

SAY "Please enter your first number"

PULL Num1

SAY "Thank you, now your second number"

PULL Num2

SAY "Well done: Now" Num1 "times" Num2 "=" Num1 \* Num2

----------------------------

Program to multiple 2 numbers

Please enter your first number

12

Thank you, now your second number

14

Well done: Now 12 times 14 = 168

The PULL Instruction is a funny little baby. What it does will depend on where you are executing your program. When running it online it will pause the execution of the routine and wait for you to type something in. When executing the REXX in JCL this instruction will try and get a line from the input queue - more about that later. For now we will worry about running it online. In the example above I typed in the numbers 12 and 14 in 'answer' to the two PULL statements. Its a pretty obvious thing to say but if you are going to use PULL like this then have a good SAY instruction just before it so you will know what to type in!

There is far more to PULL than meets the eye, you can not only accept input with it but also split it up, convert it to upper case, its very clever.

**Note: some time if we come near end of page we might need to press enter before giving value for PULL variable.**