Basic Features of the REXX Language

Objectives

Basic Features of the REXX Language

In this module, you will explore some of the components, features, and capabilities of the REXX (REstructured eXtended eXecutor) language.

You will discover how and where REXX executes, and the advantages and disadvantages of interpreted languages.

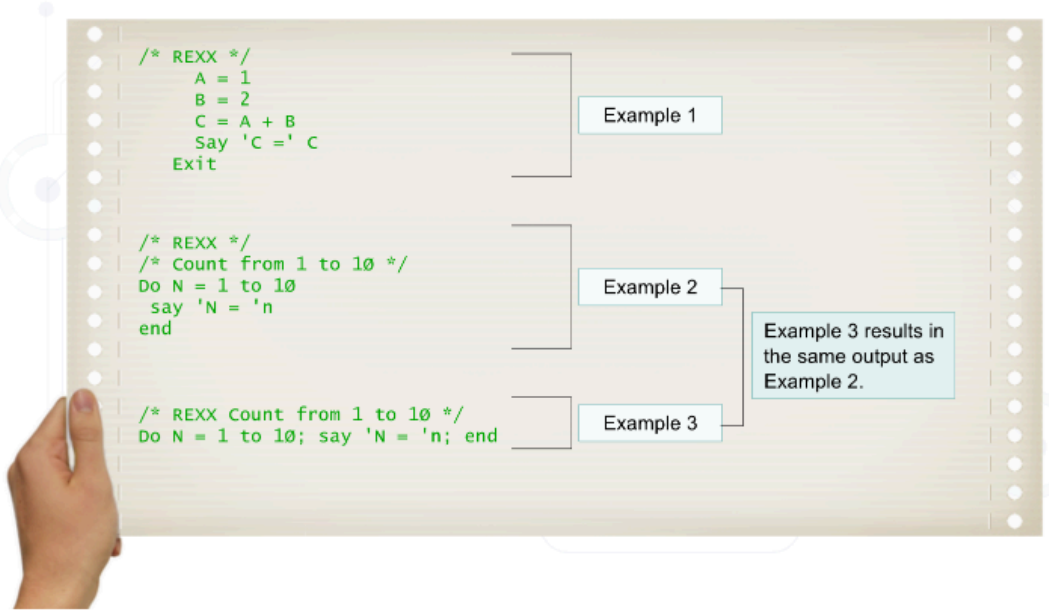
You will also examine Systems Application Architecture (SAA), and discover why the REXX language was designed to use it.

After completing this module, you will able to:

- Identify the Basic Features of the REXX Programming Language

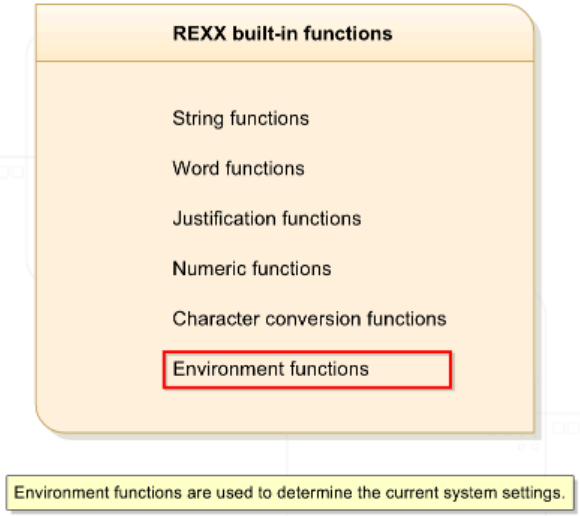
REXX is a versatile command programming language that was developed by IBM. It provides a common programming structure that has a free format and is easy to use. This makes REXX particularly good for beginners.

The basic design of REXX also makes it a very powerful language. It has a large range of built-in functions, extensive parsing capabilities, and the ability to run under any MVS address space.



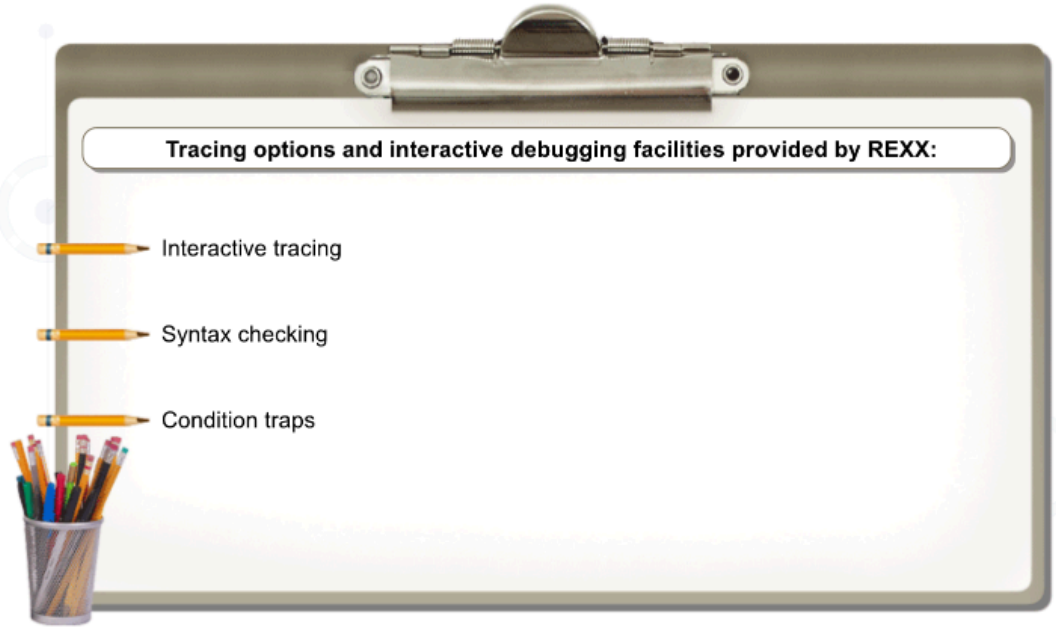
Unlike other languages, such as C++ and assembly, REXX instructions are based on English and decimal arithmetic.

This enables programs to be easily interpreted and written, even by novices.



The REXX language provides a host of built-in functions that enable you to easily perform many common actions ranging from arithmetic conversions to text formatting.

Mouse-over each function group for a brief description.



In addition to basic error messaging, REXX provides numerous tracing options and interactive debugging facilities.

REXX is an interpreted language that does not require compilation. This means that the source code of the program can be executed directly, and each source instruction is checked and "interpreted" into machine code before being executed.

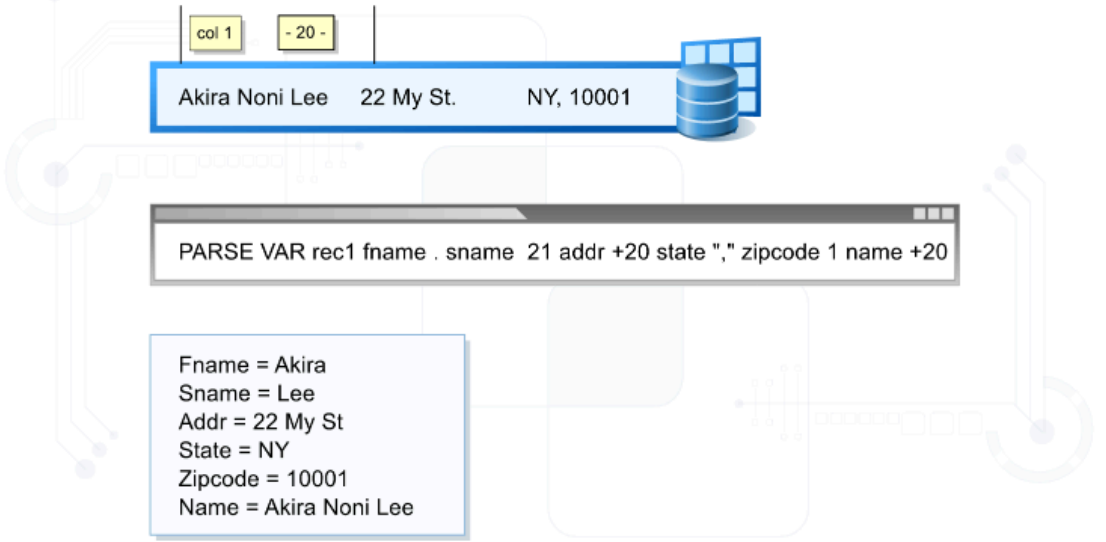
The main disadvantage of interpreted languages is that they often use up more CPU time, which makes them more expensive to run. They do, however, provide the benefits of increased flexibility and shorter development time. Although REXX does not require compilation, REXX routines can be compiled, which reduces the CPU resources required to run them, if necessary.

Click **Play** to see how an interpreted language works.

A compiled language should be considered for applications that run every day and use a lot of machine time. But if the application changes a lot, issues a large volume of commands, or has to be developed quickly, REXX is probably the best solution.

In any event, once the application is written in REXX, it can always be compiled at a later stage.

Click **Play** to see how a compiled language works.

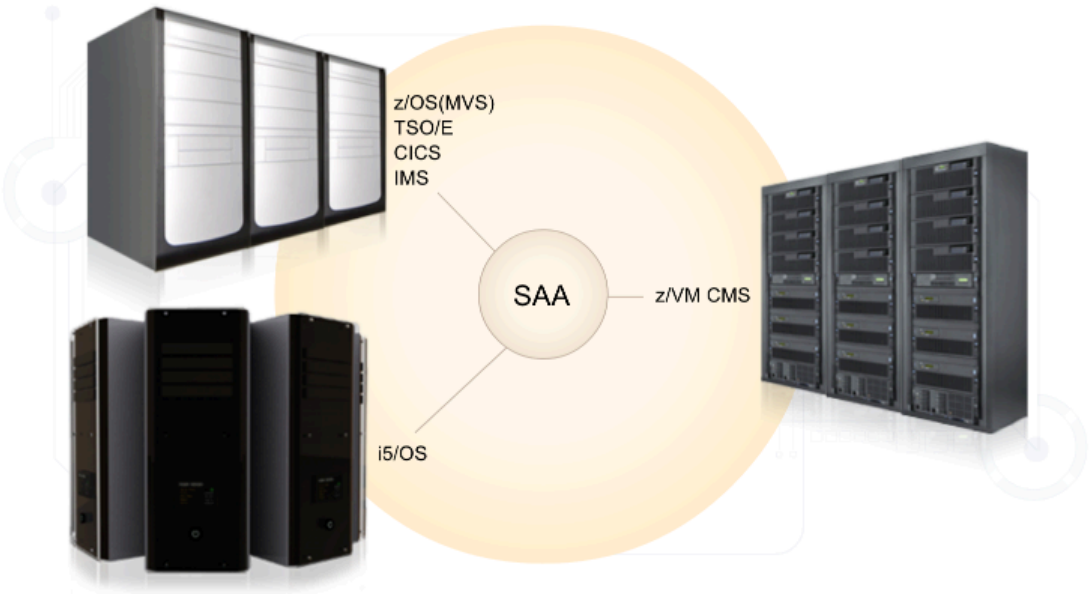


REXX includes extensive parsing capabilities for character string manipulation. "Parsing" is the act of separating computer input into its constituent parts for subsequent processing actions.

The REXX parsing facility enables a pattern to be defined and used to separate characters, numbers, and mixed input into separate variables in a single instruction.

Click **Play** to see how parsing can be used to set variables in REXX.

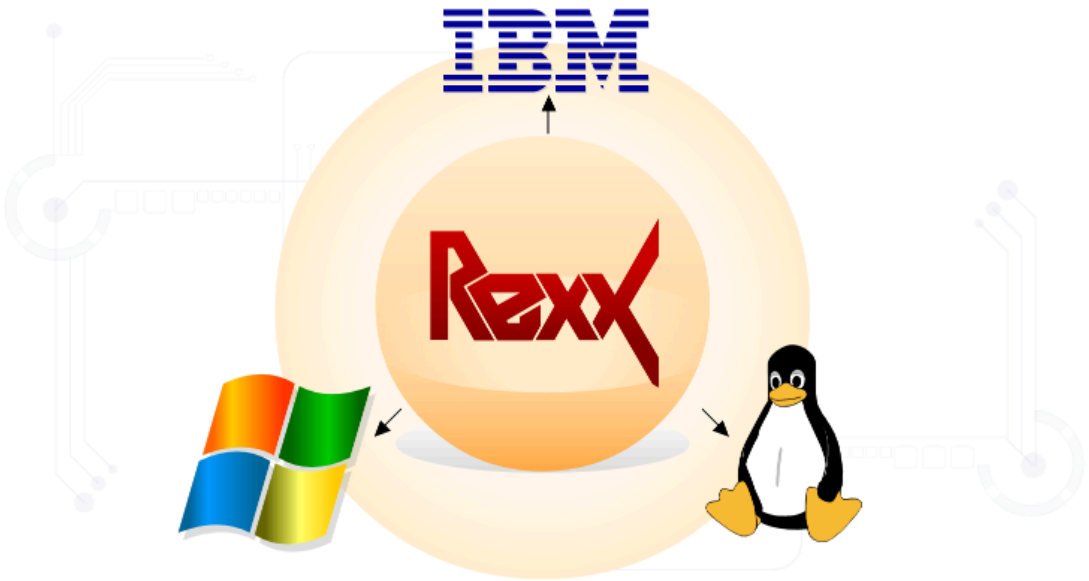
Mouse-over the five basic components of REXX TSO/E for a description of each.



When dealing with REXX programs, particularly in the IBM environment, you may encounter Systems Application Architecture (SAA).

SAA was an attempt to provide consistent interfaces across products. Many enterprise REXX systems were written to conform to SAA, which provided:

- A common programming interface
- Common communications support
- Common user access



REXX now runs on various IBM and non-IBM platforms, including z/OS, z/VM, i5/OS, Microsoft Windows, and Linux as an integrated part of the operating system or as an add-on product.

REXX acts on all these platforms as both a powerful scripting language to drive operating system functions and as a complete programming language.