# 3) Parsing

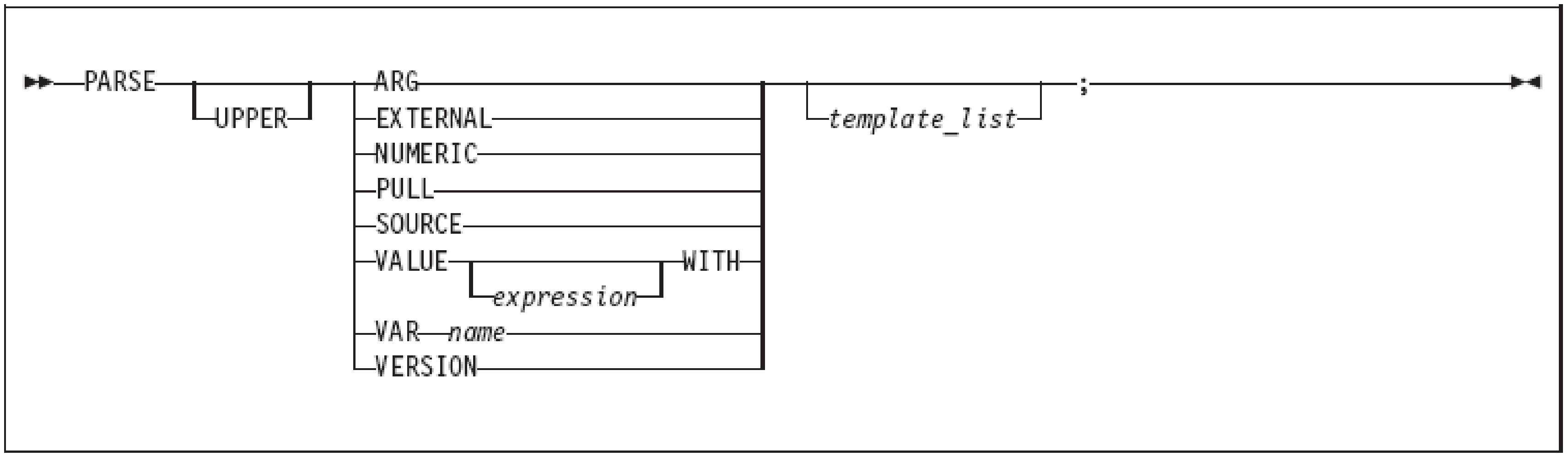
Instructions: PARSE, ARG, Patterns.

Resources: TSO REXX Reference
Chapter 5. Parsing

This course has been prepared by Milos Forman for MCoE needs only!

# Parse instruction

## PARSE



PARSE assigns data (from various sources) to one or more variables according to the rules of parsing (see Chapter 5, “Parsing,” on page 165).

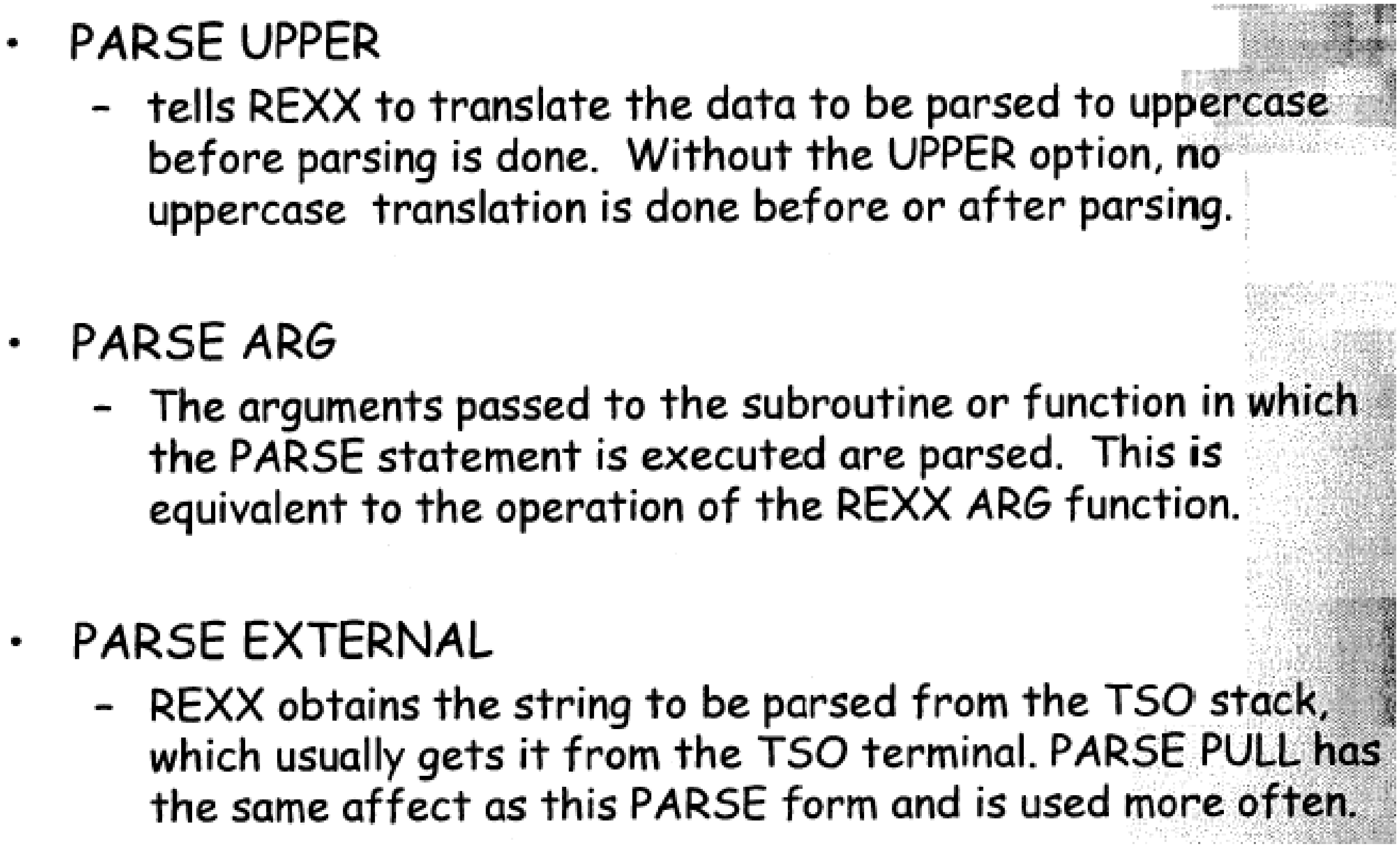
# Parse instruction

The PARSE instruction tells REXX how to assign data to one or more variables. The data to assign can be from the terminal, the data stack, or from arguments passed to a subroutine or function. The way in which REXX assigns data to a variable is governed by what is known as a 'parsing template', discussed below.

'template' is made up of alternating optional "patterns" and variable names. "patterns" are of two types: those that cause parsing to search for a matching string (variable patterns and literal patterns) and numeric patterns that supply a string position number in the data from which parsing is to extract data. Any number of "patterns" and

variables can be intermixed.

# Parse - Operands



- PARSE UPPER
  - tells REXX to translate the data to be parsed to uppercase before parsing is done. Without the UPPER option, no uppercase translation is done before or after parsing.
- PARSE ARG
  - The arguments passed to the subroutine or function in which the PARSE statement is executed are parsed. This is equivalent to the operation of the REXX ARG function.
- PARSE EXTERNAL
  - REXX obtains the string to be parsed from the TSO stack, which usually gets it from the TSO terminal. PARSE PULL has the same affect as this PARSE form and is used more often.

## Parse - Operands

- PARSE NUMERIC
  - returns the current settings for the NUMERIC options DIGITS, FUZZ, and FORM, in that order.

- PARSE PULL
  - This form of the PARSE instruction makes REXX get the next string from the REXX data stack. If the stack is empty, REXX will get the string from TSO terminal.

- PARSE VALUE
  - this PARSE form parses a string under the control of the parsing template, described previously.

# Parse - Operands

**PARSE SOURCE**

parses data describing the source of the program running. The language processor returns a string that is fixed (does not change) while the program is running.

**PARSE VAR** *name*

parses the value of the variable *name*. The *name* must be a symbol that is valid as a variable name (that is, it cannot start with a period or a digit). Note that the variable *name* is not changed unless it appears in the template, so that for

**PARSE VERSION**

parses information describing the language level and the date of the language processor. This information consists of five blank-delimited words:

## Parse - ARG

- Syntax
  - PARSE UPPER ARG [template]
- Parses the arguments passed to the program according to the template, optionally first translating it to uppercase
- Shortened to - ARG

```
ARG test_word
SAY "You have passed the program : "test_word
```

## Parse - VAR

- Syntax
  - PARSE VAR name [template]

```
ARG test_words
PARSE VAR test_words first_word second_word left_overs
SAY "You have passed the program : "first_word
SAY "You have passed the program : "second_word
SAY "You have passed the program : "left_overs
```

```
You have passed the program : THIS
You have passed the program : IS
You have passed the program : A TEST OF WORDS
***
```

## Parse - PULL

- Syntax
  - PARSE PULL [template]

```
SAY "Please enter your first name."
PARSE PULL first_name
```

## Parse – EXTERNAL

- Syntax
  - PARSE EXTERNAL [template]

```
SAY "Please enter your first name."
PARSE EXTERNAL first_name
```

## Parse – VALUE WITH

- Syntax
  - PARSE VALUE [expression] WITH [template]

- Use VALUE WITH to break apart long strings or expressions that need to be evaluated first

```
new_date = "12/11/2000"
PARSE VALUE new_date WITH mm "/" dd "/" yyyy
SAY mm
SAY dd
SAY yyyy
```

## Parse - Templates

- Simplest form consists of a list of variable names
- String being parsed is split into words
- each word is assigned to a variable from left to right
- The last variable is assigned whatever is left.

```
name = "Mike Fred Bob Jones"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

## Parse - Templates

- If there are fewer words than variables, any remaining variables are assigned the null string.
- Leading blanks are removed from each word.

```
name = "Mike"
PARSE VALUE name WITH first_name left_cvers
SAY first_name
SAY left_overs
```

## Parse - Templates

- Example - What happens here ?

```
name = "Mike Fred Jones"
PARSE VAR name new_name name
SAY name
SAY new_name
```

# Parse – Literal Paterns

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names one "," two "," three
SAY one
SAY two
SAY three
```

```
Mike
Fred
Joe
***
```

# Parse – Using Placeholders

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names . "," . "," last_name
SAY last_name
```

```
Joe
***
```

## Parse – Positional Paterns

- Example

```
names = "Some text, to be, split up!"
PARSE VAR names one 10 two 20 three
SAY one
SAY two
SAY three
```

```
Some text
, to be, s
plit up!
***
```

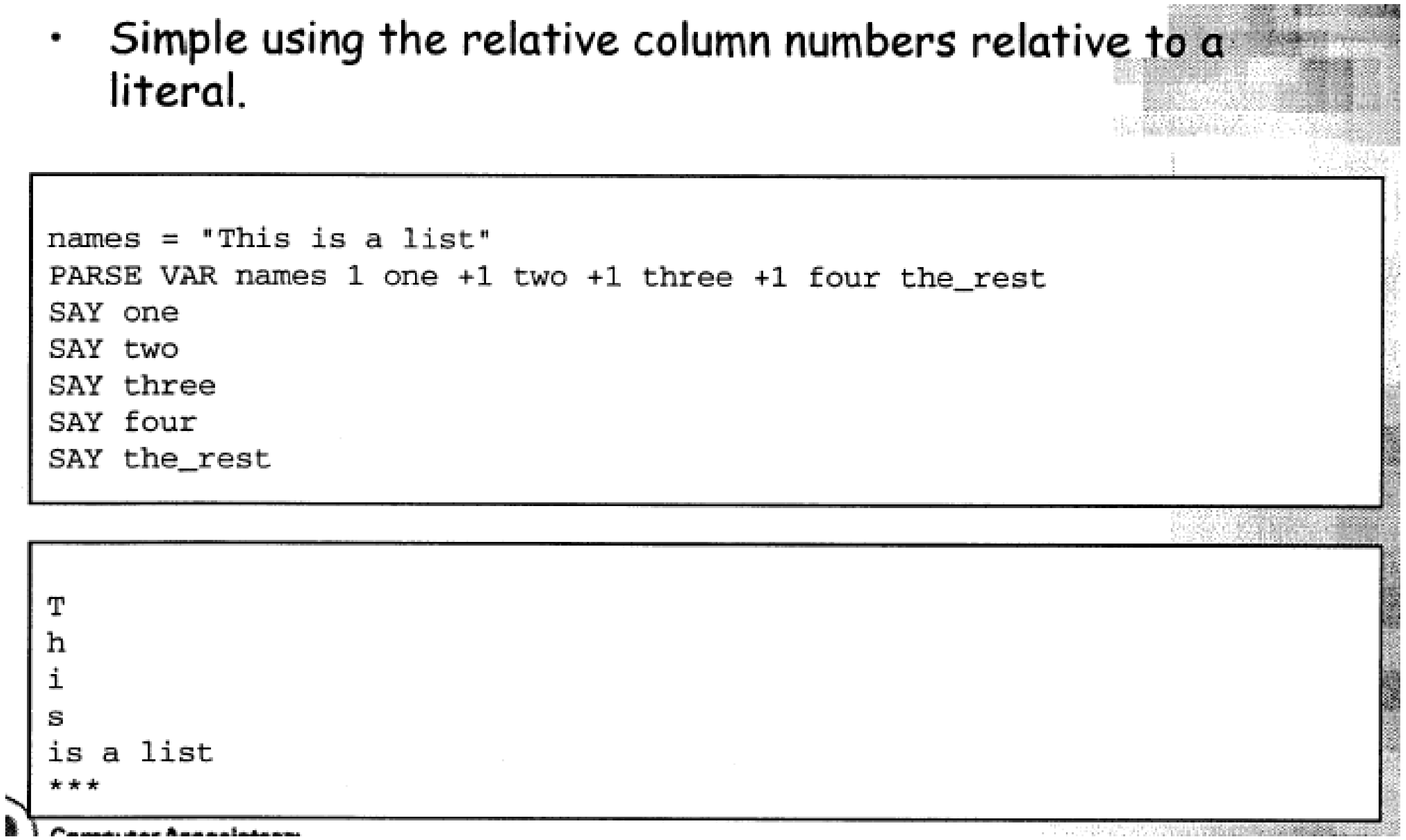
## Parse – Relative Positional Paterns

- Example

```
names = "0987654321"
PARSE VAR names 3 one +2 two +4 three 2 four
SAY one
SAY two
SAY three
SAY four
```

```
87
6543
21
987654321
***
```

# Parse – Sample Relative Positional Paterns



- Simple using the relative column numbers relative to a literal.

```
names = "This is a list"
PARSE VAR names 1 one +1 two +1 three +1 four the_rest
SAY one
SAY two
SAY three
SAY four
SAY the_rest
```

```
T
h
i
s
is a list
***
```

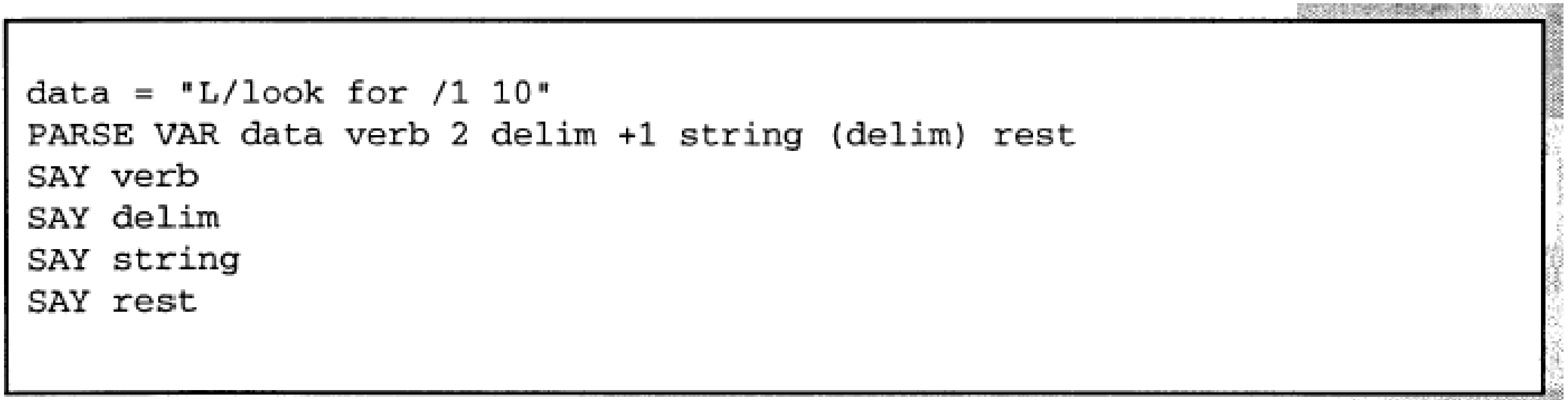
Computer Associates

# Parse – Variable Paterns

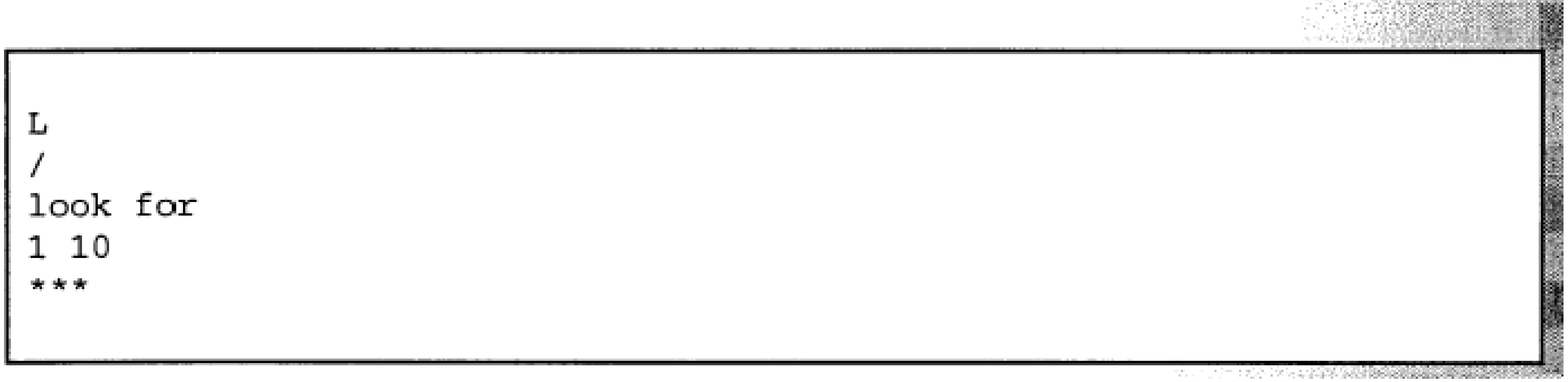
- Specify a pattern by using the value of a variable instead of a fixed string number
- Place the name of the variable to be used as the pattern in parentheses
- If a +, - or = sign precedes the parentheses, the value of the variable is then used as though it were a relative column number'

```
name = "Mike"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

# Parse – Variable Paterns Example

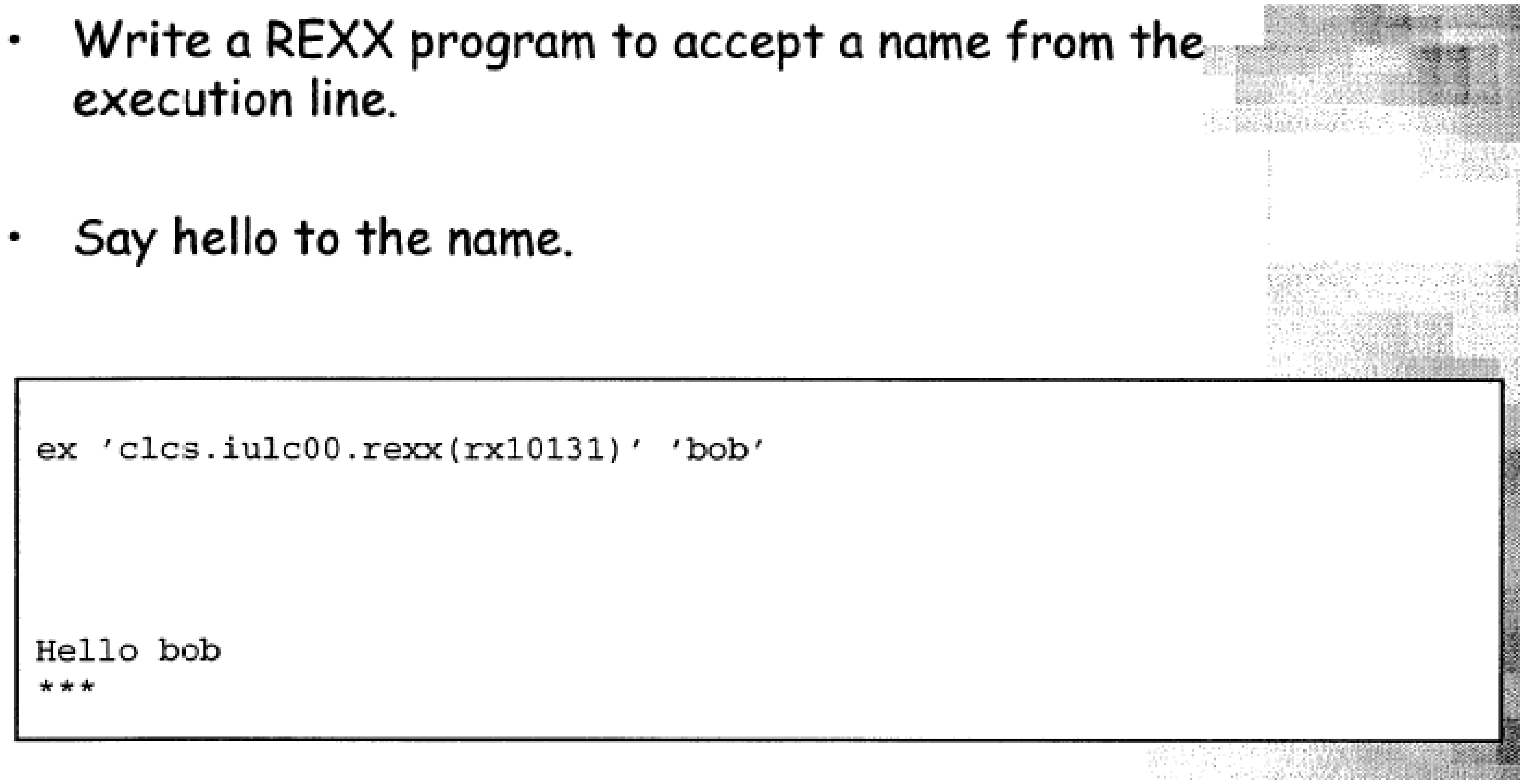


```
data = "L/look for /1 10"
PARSE VAR data verb 2 delim +1 string (delim) rest
SAY verb
SAY delim
SAY string
SAY rest
```



```
L
/
look for
1 10
***
```

## Work Section 3.1



- Write a REXX program to accept a name from the execution line.
- Say hello to the name.

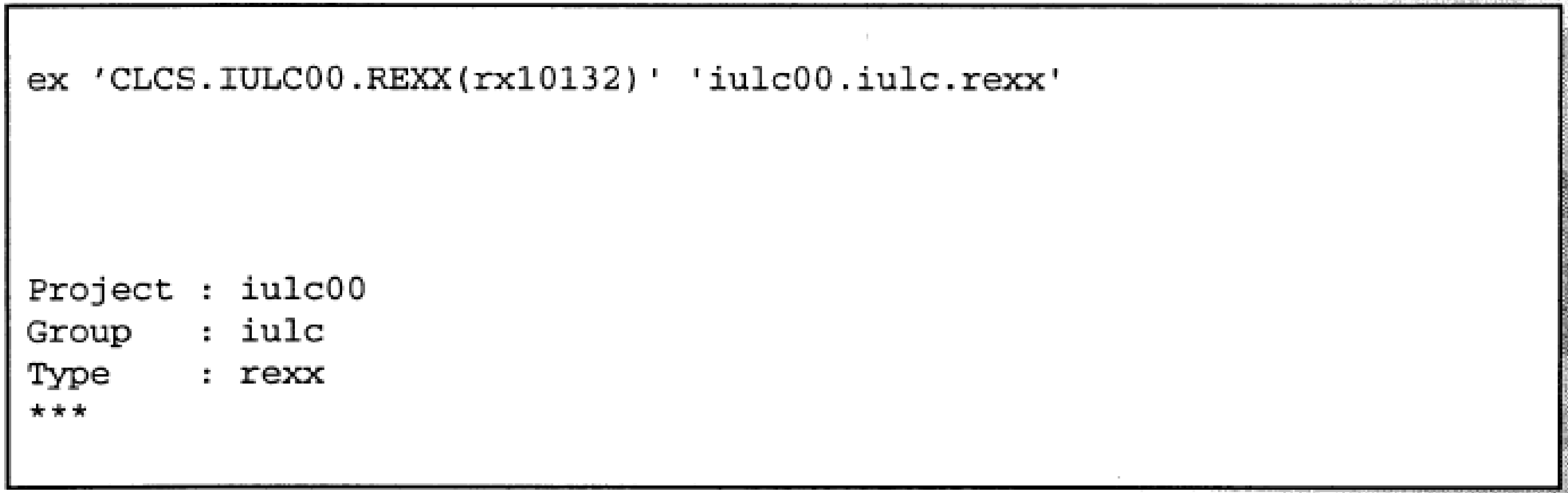
```
ex 'clcs.iulc00.rexx(rx10131)' 'bob'



Hello bob
***
```

# Work Section 3.2

- Write a REXX program to accept a 3 level qualified dataset from the execution line.
- Then display each section to the screen



```
ex 'CLCS.IULC00.REXX(rx10132)' 'iulc00.iulc.rexx'




Project : iulc00
Group   : iulc
Type    : rexx
***
```

## Additional Programs

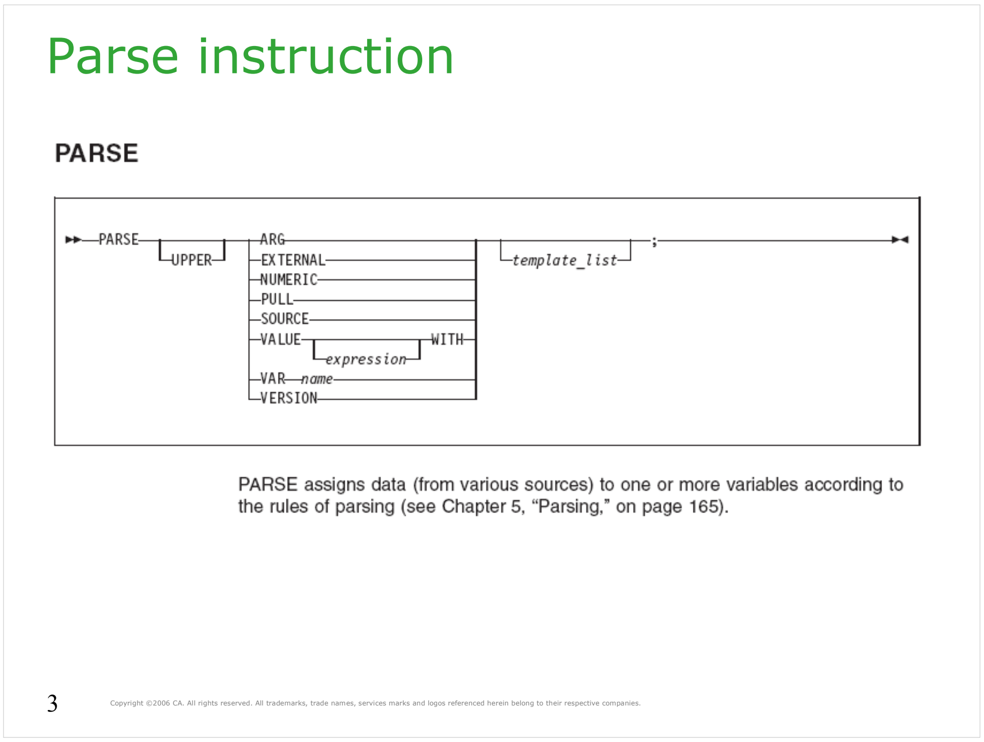
- If you had proceeding spaces in work section 1.2 then re-write using PARSE to remove the spaces.

## 3) Parsing

Instructions: PARSE, ARG,
Patterns.

Resources:TSO REXX Reference
Chapter 5. Parsing

This course has been prepared by Milos Forman for MCoE needs only!



## Parse instruction

**PARSE**

```
>>──PARSE──┬───────┬──┬─ARG──────────────────────────────┬──┬───────────────┬──;──────────────><
           └─UPPER─┘  ├─EXTERNAL─────────────────────────┤  └─template_list─┘
                      ├─NUMERIC──────────────────────────┤
                      ├─PULL─────────────────────────────┤
                      ├─SOURCE───────────────────────────┤
                      ├─VALUE──┬──────────────┬──WITH────┤
                      │        └─expression───┘          │
                      ├─VAR──name────────────────────────┤
                      └─VERSION──────────────────────────┘
```

PARSE assigns data (from various sources) to one or more variables according to the rules of parsing (see Chapter 5, "Parsing," on page 165).

3

The parse instruction is used to assign data from various sources to one or more variables.

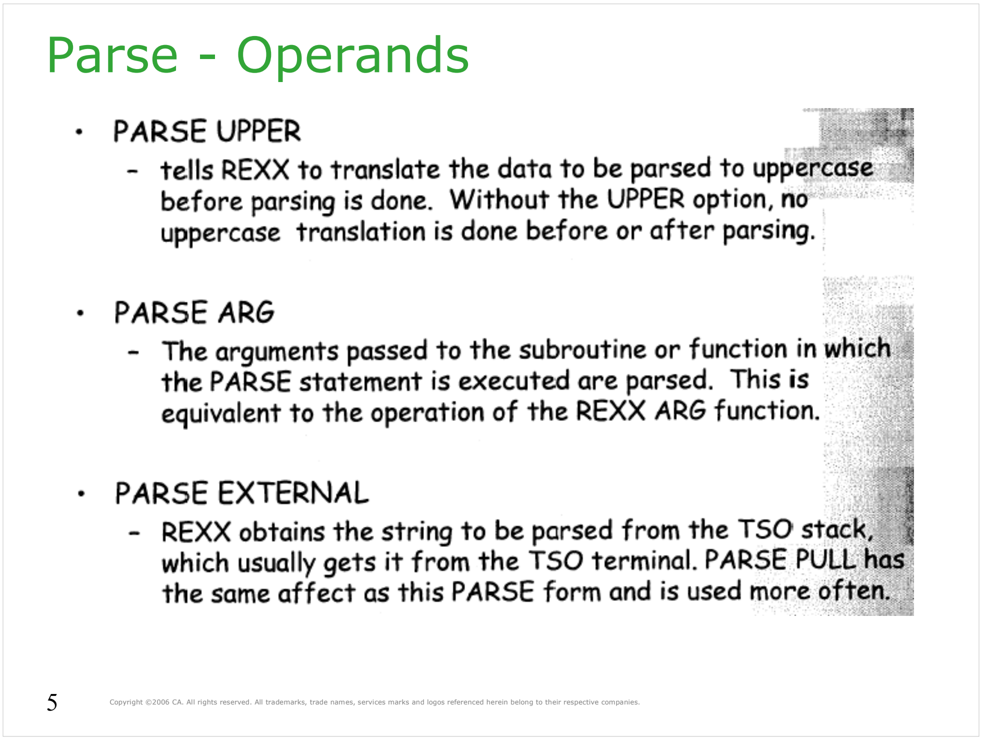
See description  on next slide.

## Parse instruction

The PARSE instruction tells REXX how to assign data to one or more variables. The data to assign can be from the terminal, the data stack, or from arguments passed to a subroutine or function. The way in which REXX assigns data to a variable is governed by what is known as a 'parsing template', discussed below.

'template' is made up of alternating optional "patterns" and variable names. "patterns" are of two types: those that cause parsing to search for a matching string (variable patterns and literal patterns) and numeric patterns that supply a string position number in the data from which parsing is to extract data. Any number of "patterns" and

variables can be intermixed.



## Parse - Operands

- PARSE UPPER
  - tells REXX to translate the data to be parsed to uppercase before parsing is done. Without the UPPER option, no uppercase translation is done before or after parsing.
- PARSE ARG
  - The arguments passed to the subroutine or function in which the PARSE statement is executed are parsed. This is equivalent to the operation of the REXX ARG function.
- PARSE EXTERNAL
  - REXX obtains the string to be parsed from the TSO stack, which usually gets it from the TSO terminal. PARSE PULL has the same affect as this PARSE form and is used more often.

5

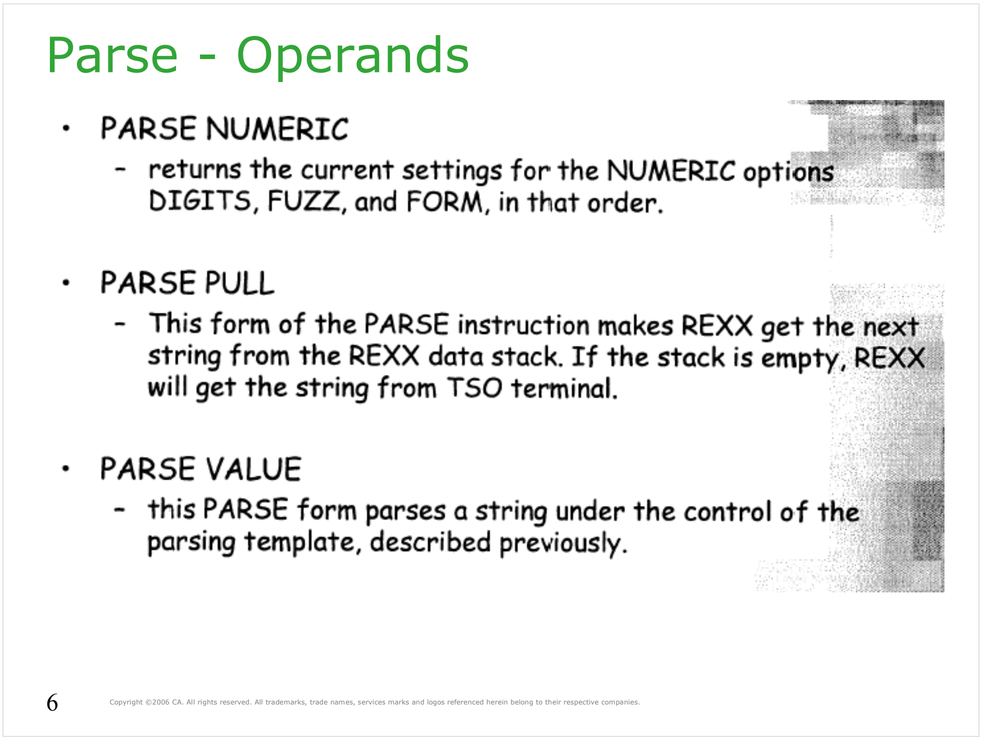
**PARSE** assigns data from various sources to one or more variables according to the rules of parsing.

**PARSE ARG** parses the string passed to a program or internal routine as input arguments.

**PARSE EXTERNAL** If you can, use PARSE PULL instead of PARSE EXTERNAL.

Invoke the example from option 6 COMMAND:

exec ,mcoe.rexa.rexx(rx20133)' 'Passed argument,



## Parse - Operands

- PARSE NUMERIC
  - returns the current settings for the NUMERIC options DIGITS, FUZZ, and FORM, in that order.
- PARSE PULL
  - This form of the PARSE instruction makes REXX get the next string from the REXX data stack. If the stack is empty, REXX will get the string from TSO terminal.
- PARSE VALUE
  - this PARSE form parses a string under the control of the parsing template, described previously.

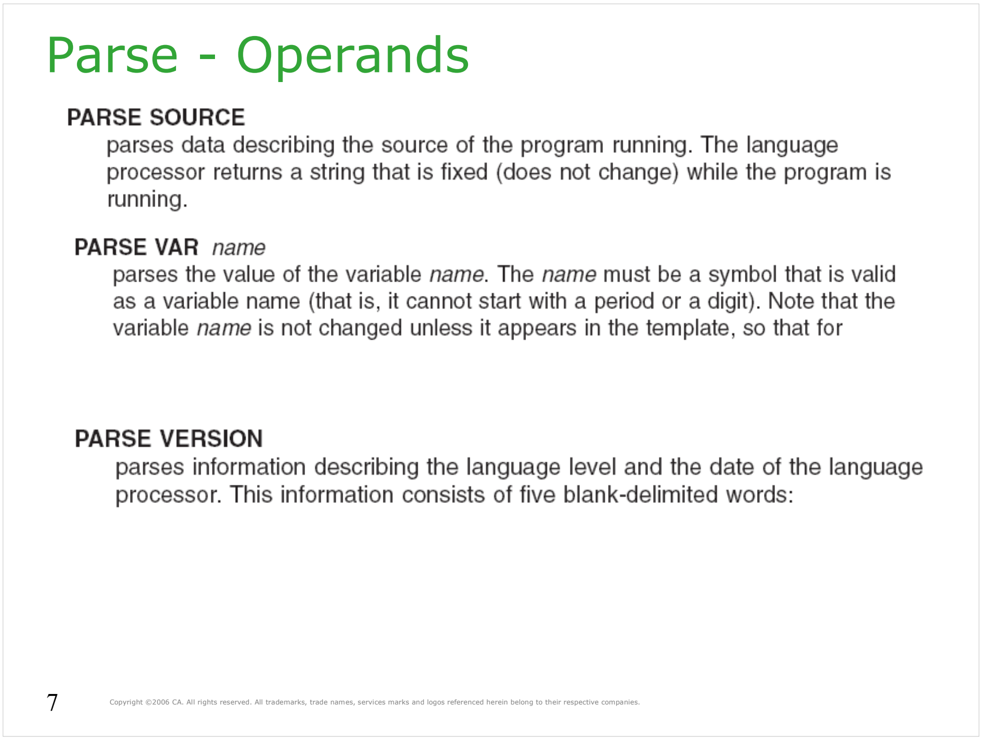
6

**PARSE NUMERIC** returns the current settings for the NUMERIC options DIGITS, FUZZ, and FORM.

**PARSE PULL** parses the next string from the external data queue. If the external data queue is empty, PARSE PULL reads a line from the default input stream - the user’s terminal.

**PARSE VALUE** parses the data that is the result of evaluating *expression*.

See ,MCOE.REXA.REXX(RX20134),



## Parse - Operands

**PARSE SOURCE**
parses data describing the source of the program running. The language processor returns a string that is fixed (does not change) while the program is running.

**PARSE VAR** *name*
parses the value of the variable *name*. The *name* must be a symbol that is valid as a variable name (that is, it cannot start with a period or a digit). Note that the variable *name* is not changed unless it appears in the template, so that for

**PARSE VERSION**
parses information describing the language level and the date of the language processor. This information consists of five blank-delimited words:

7

**PARSE SOURCE** parses data describing the source of the program running.

The source string contains the following tokens:

1. The characters TSO.
2. The string COMMAND, FUNCTION, or SUBROUTINE.
3. Usually, name of the exec in uppercase.
4. Name of the DD from which the exec was loaded.
5. Name of the data set from which the exec was loaded.
6. Name of the exec as it was called.

And some others. I have never used it.

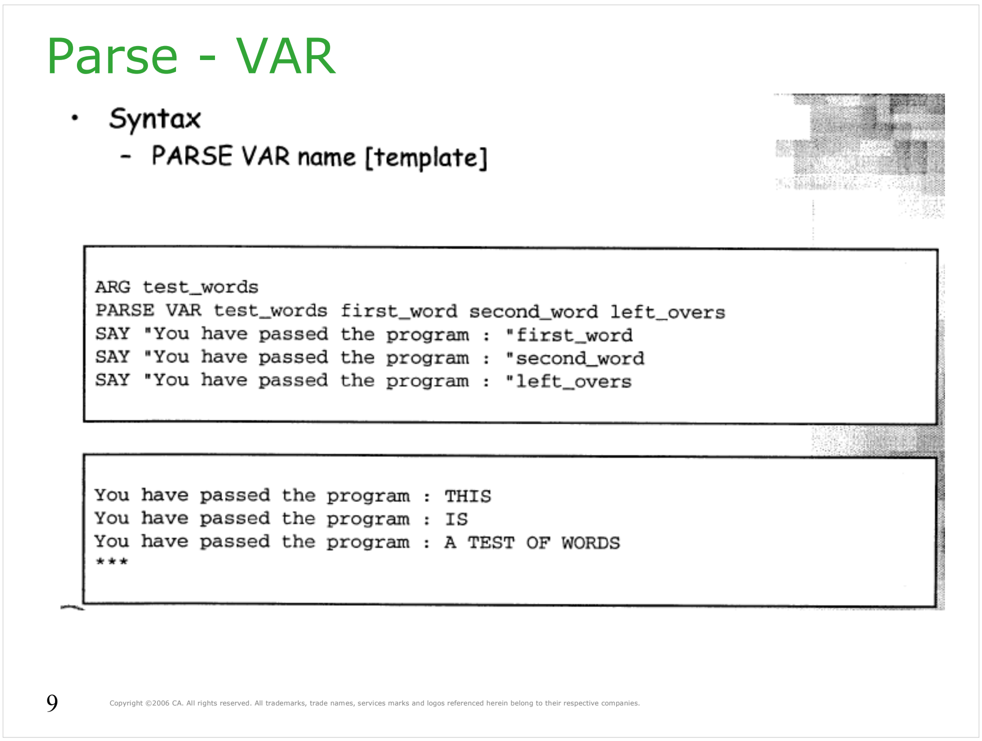
**PARSE VAR** *name* parses the value of the variable *name*.

**PARSE VERSION** parses information describing the language level and the date of the language processor.

## Parse - ARG

- Syntax
  - PARSE UPPER ARG [template]
- Parses the arguments passed to the program according to the template, optionally first translating it to uppercase
- Shortened to - ARG

```
ARG test_word
SAY "You have passed the program : "test_word
```



## Parse - VAR

- Syntax
  - PARSE VAR name [template]

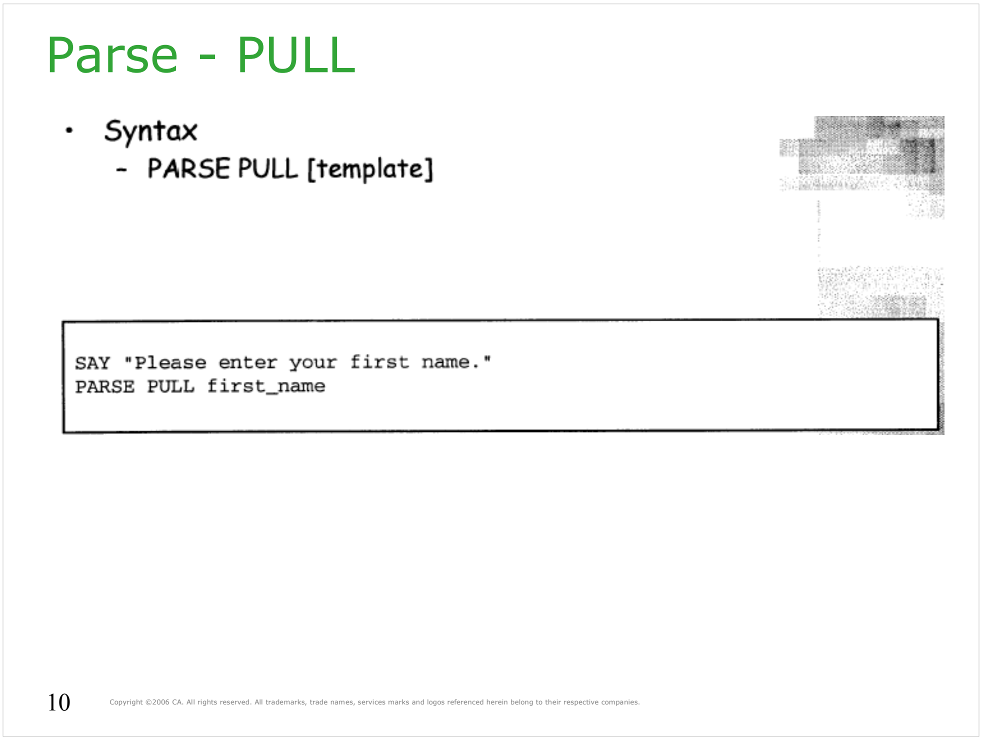
```
ARG test_words
PARSE VAR test_words first_word second_word left_overs
SAY "You have passed the program : "first_word
SAY "You have passed the program : "second_word
SAY "You have passed the program : "left_overs
```

```
You have passed the program : THIS
You have passed the program : IS
You have passed the program : A TEST OF WORDS
***
```

9

Invoke the example from option 6 COMMAND:

exec ,mcoe.rexa.rexx(rx20136)' 'This is a test of words,



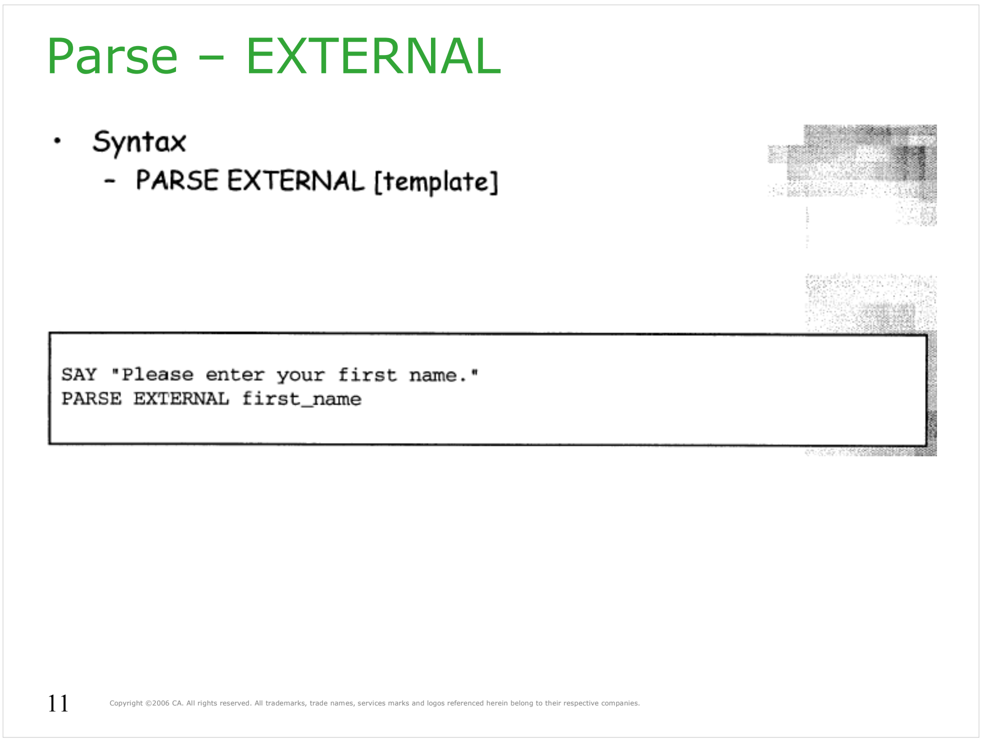
## Parse - PULL

- Syntax
  - PARSE PULL [template]

```
SAY "Please enter your first name."
PARSE PULL first_name
```

10

You can use similar instruction PULL (instead of PARSE PULL), which translates string to uppercase.



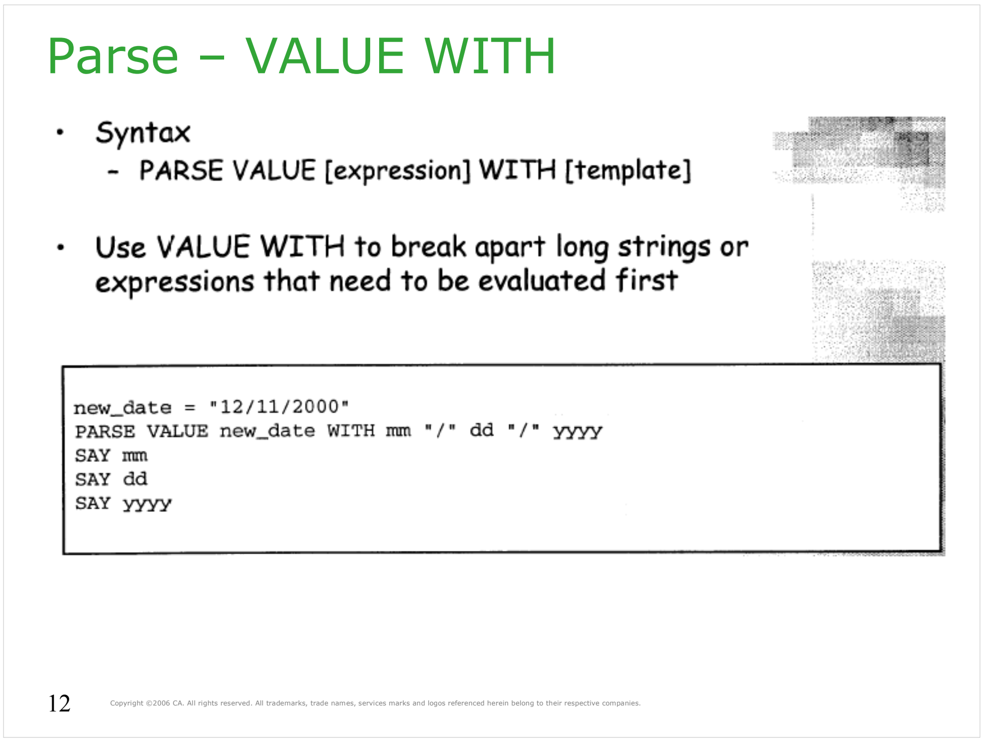
## Parse – EXTERNAL

- Syntax
  - PARSE EXTERNAL [template]

```
SAY "Please enter your first name."
PARSE EXTERNAL first_name
```

11

TSO and VM environments ONLY. It is recommended to use PARSE PULL instead of PARSE EXTERNAL.



## Parse – VALUE WITH

- Syntax
  - PARSE VALUE [expression] WITH [template]
- Use VALUE WITH to break apart long strings or expressions that need to be evaluated first

```
new_date = "12/11/2000"
PARSE VALUE new_date WITH mm "/" dd "/" yyyy
SAY mm
SAY dd
SAY yyyy
```

12

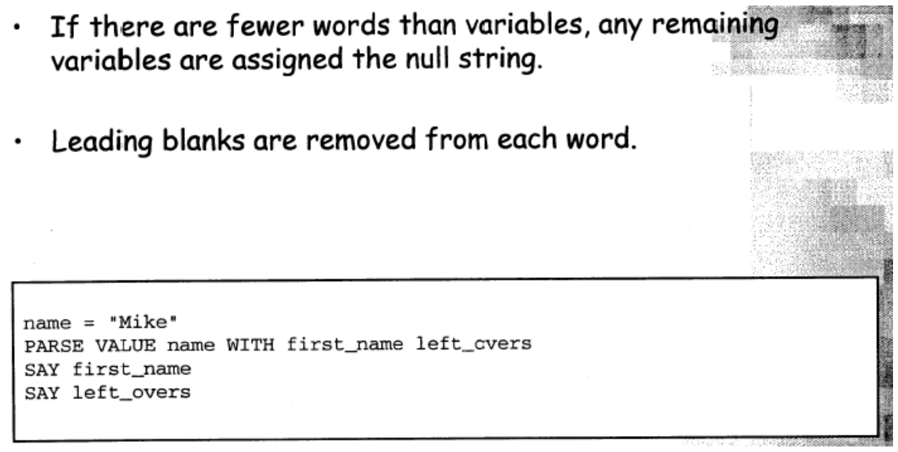
See ‘MCOE.REXA.REXX(RX20139)’

## Parse - Templates

- Simplest form consists of a list of variable names
- String being parsed is split into words
- each word is assigned to a variable from left to right
- The last variable is assigned whatever is left.

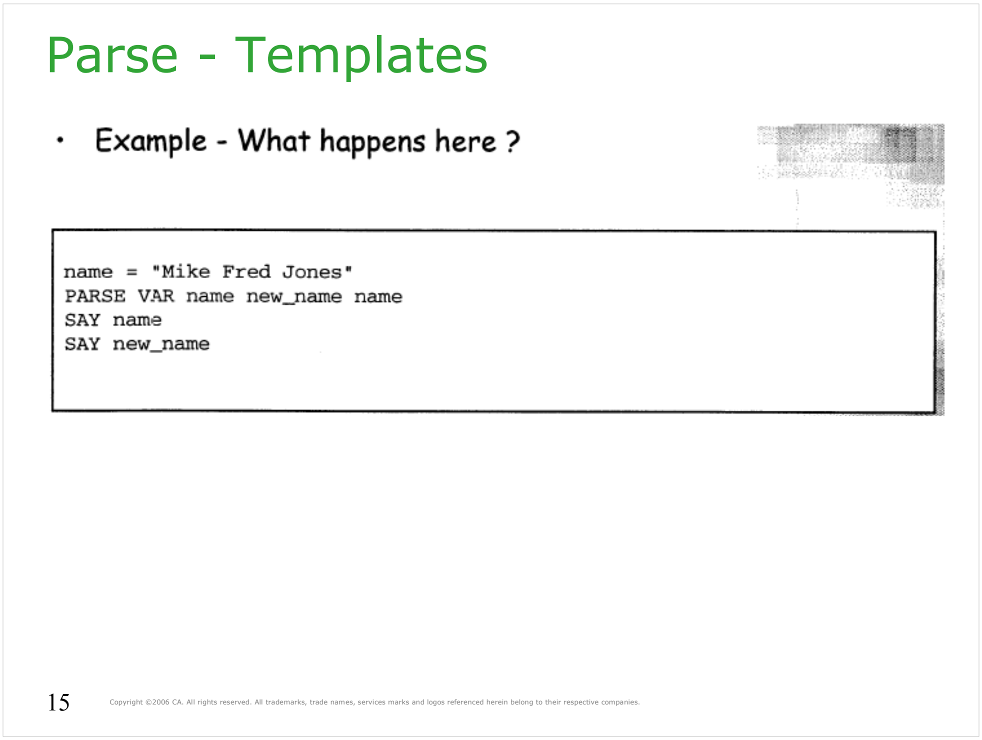
```
name = "Mike Fred Bob Jones"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

## Parse - Templates



- If there are fewer words than variables, any remaining variables are assigned the null string.
- Leading blanks are removed from each word.

```
name = "Mike"
PARSE VALUE name WITH first_name left_cvers
SAY first_name
SAY left_overs
```



## Parse - Templates

- Example - What happens here ?

```
name = "Mike Fred Jones"
PARSE VAR name new_name name
SAY name
SAY new_name
```

15

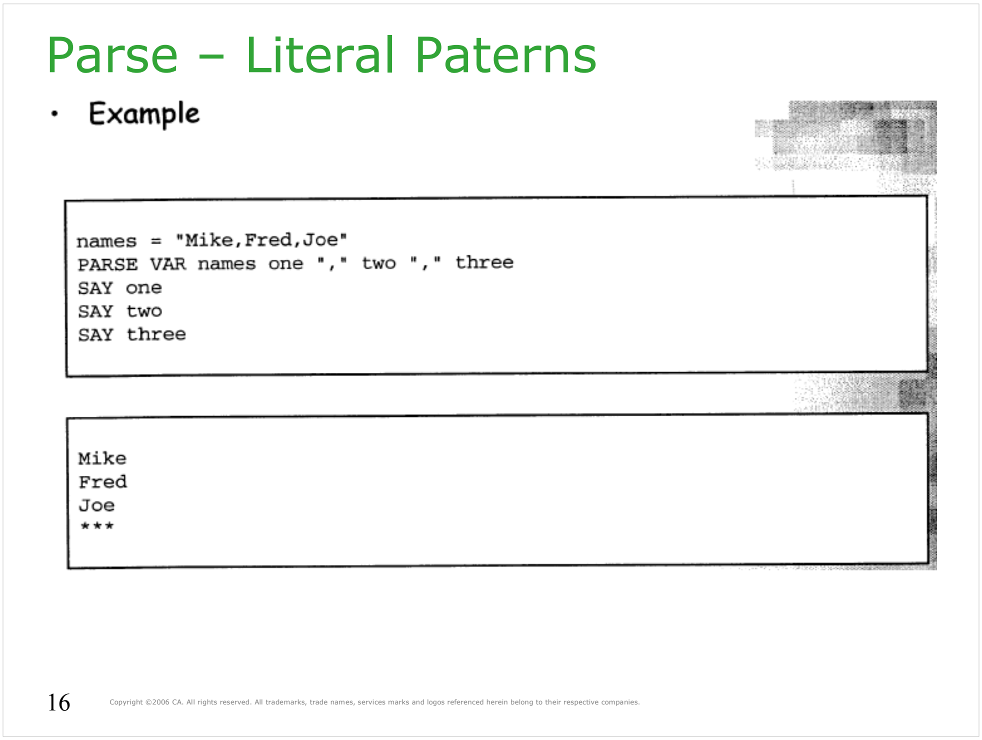
**PARSE VAR *name*** parses the value of the variable ***name***, so for example:

PARSE VAR   name   new_name   name

removes the first word from ***name***, puts it in the variable ***new_name***, and assigns the remainder back to ***name***.

So the output from the example is:

```
Fred Jones
Mike
```



## Parse – Literal Paterns

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names one "," two "," three
SAY one
SAY two
SAY three
```

```
Mike
Fred
Joe
***
```

16

**PARSE VAR *names*** parses the value of the variable ***names***, so for example:

PARSE VAR   names   one   “,”   two   “,”   three

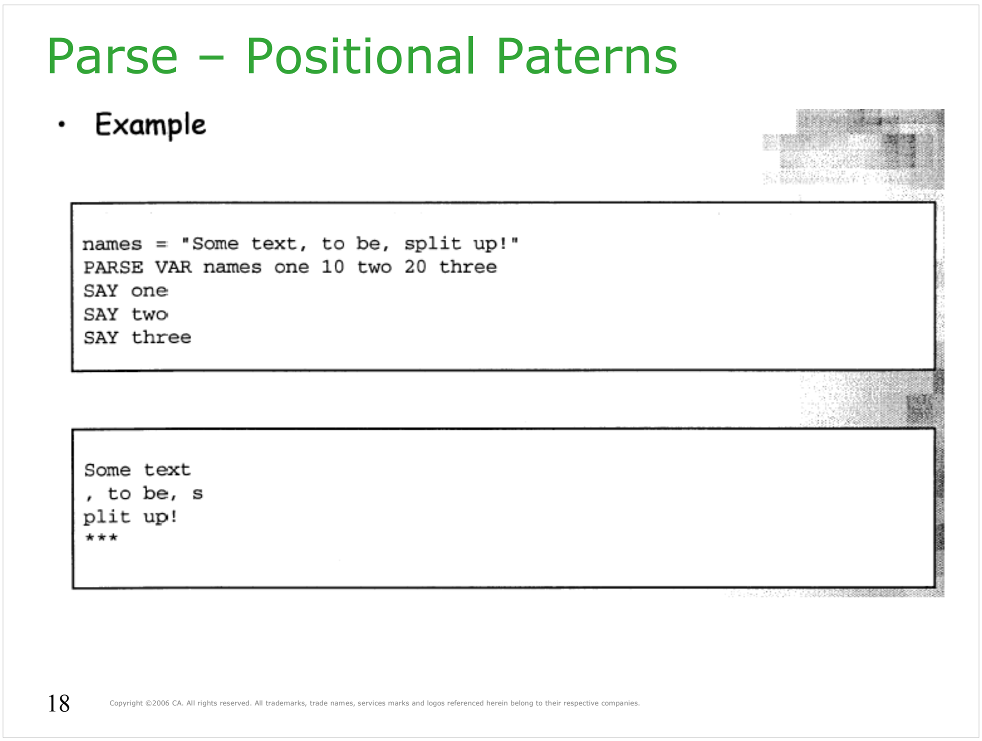
removes the first word from ***names***, puts it in the variable ***one***, and assigns the remainder back to ***names*** *and similarly for the rest of* ***names*** *variable*.

## Parse – Using Placeholders

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names . "," . "," last_name
SAY last_name
```

```
Joe
***
```



## Parse – Positional Paterns

- Example

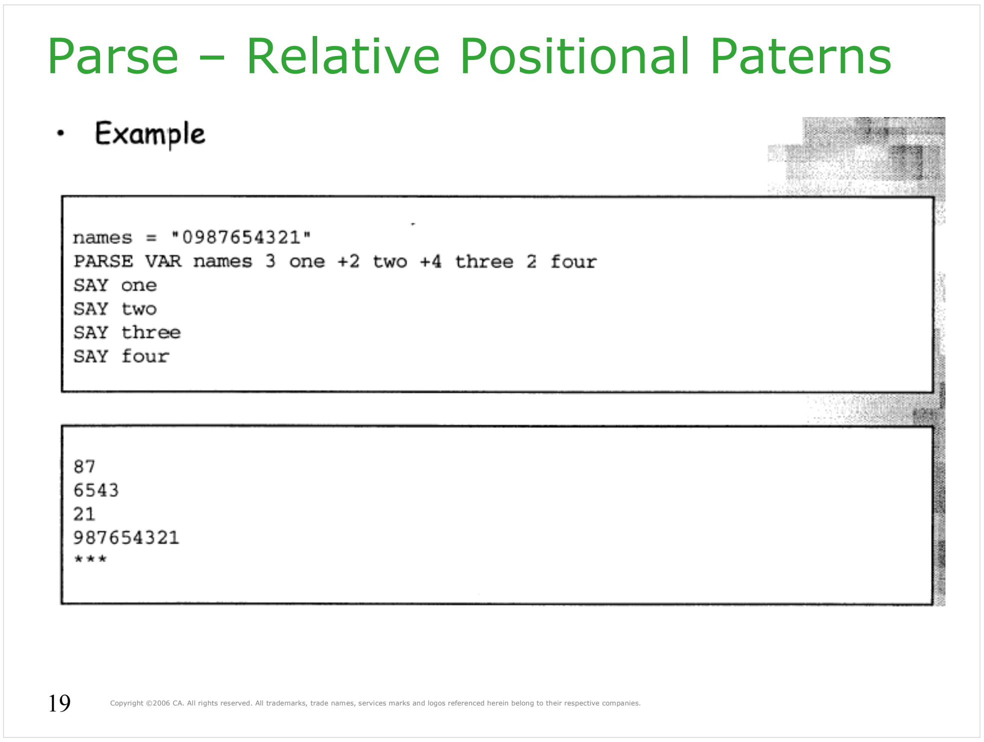
```
names = "Some text, to be, split up!"
PARSE VAR names one 10 two 20 three
SAY one
SAY two
SAY three
```

```
Some text
, to be, s
plit up!
***
```

18

Digits 10 and 20 indicate a starting column.

See ‘MCOE.REXA.REXX(RX201315)’



## Parse – Relative Positional Paterns

- Example

```
names = "0987654321"
PARSE VAR names 3 one +2 two +4 three 2 four
SAY one
SAY two
SAY three
SAY four
```

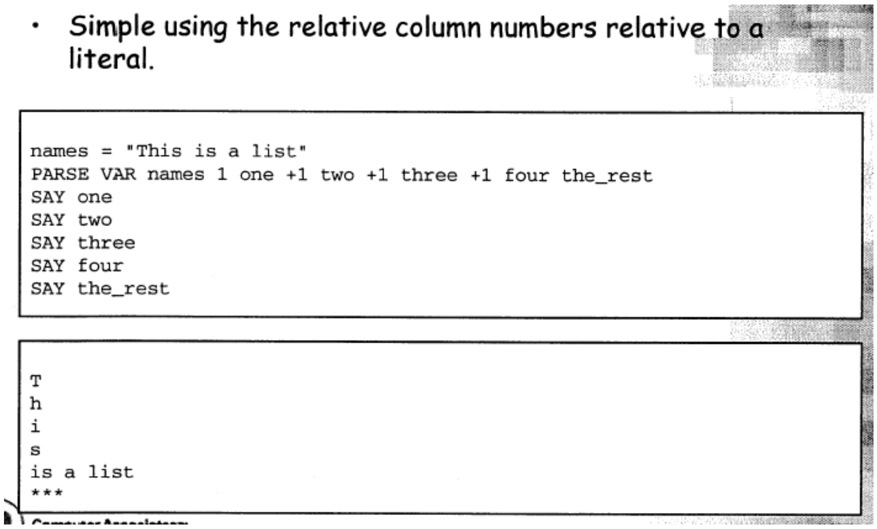
```
87
6543
21
987654321
***
```

19

Digits 3 and 2 indicate a starting column.

Digits +2 and +4 indicate relative starting column.

## Parse – Sample Relative Positional Paterns

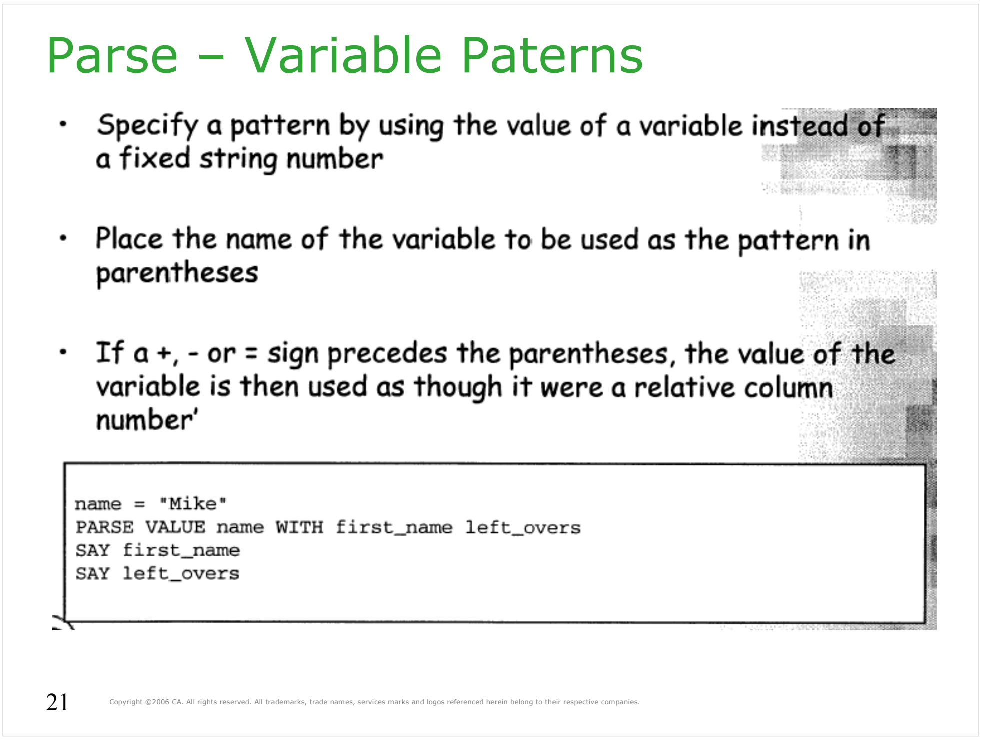


- Simple using the relative column numbers relative to a literal.

```
names = "This is a list"
PARSE VAR names 1 one +1 two +1 three +1 four the_rest
SAY one
SAY two
SAY three
SAY four
SAY the_rest
```

```
T
h
i
s
is a list
***
```

Write it and test it.



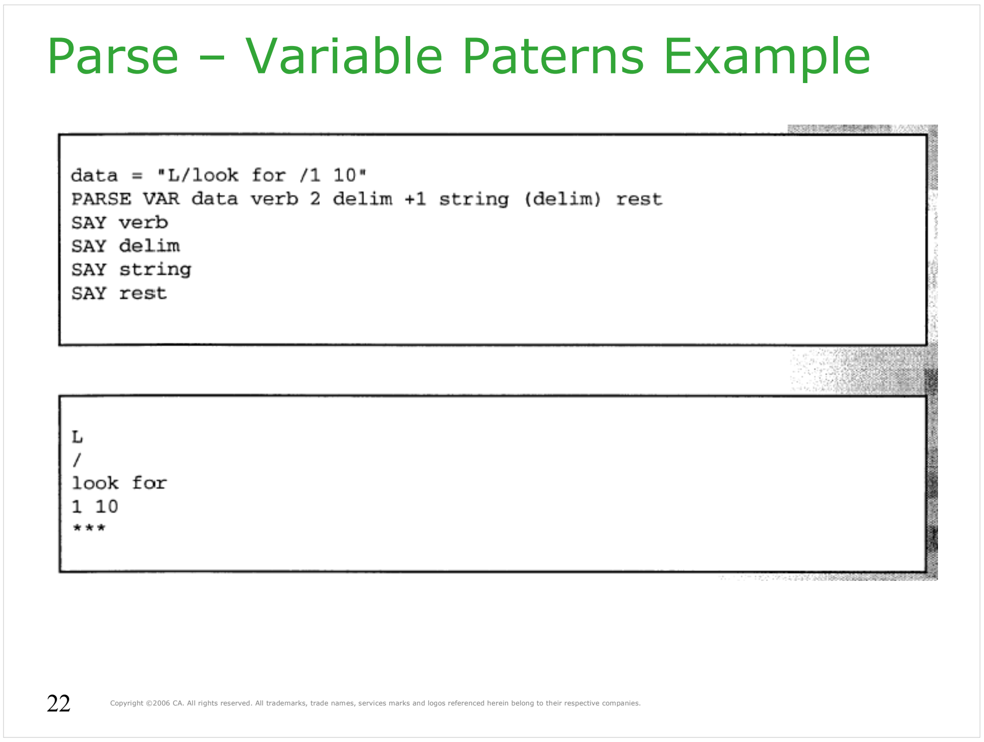
## Parse – Variable Paterns

- Specify a pattern by using the value of a variable instead of a fixed string number
- Place the name of the variable to be used as the pattern in parentheses
- If a +, - or = sign precedes the parentheses, the value of the variable is then used as though it were a relative column number'

```
name = "Mike"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

21

It was discussed on slides 12 and 13.



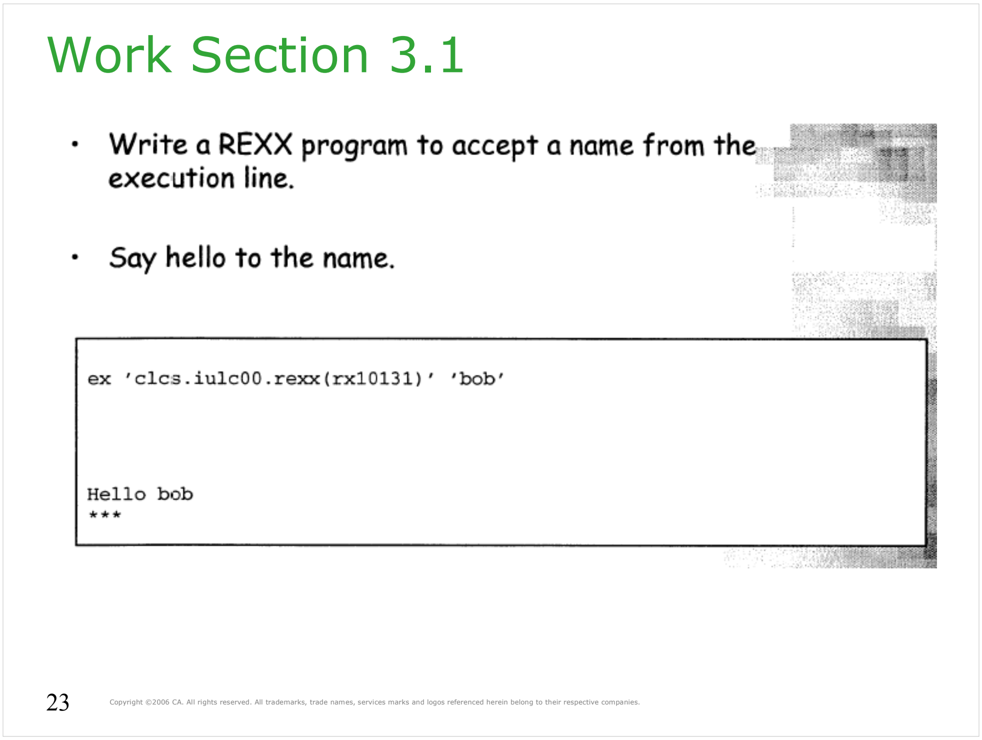
## Parse – Variable Paterns Example

```
data = "L/look for /1 10"
PARSE VAR data verb 2 delim +1 string (delim) rest
SAY verb
SAY delim
SAY string
SAY rest
```

```
L
/
look for
1 10
***
```

22

We discussed it before.



## Work Section 3.1

- Write a REXX program to accept a name from the execution line.
- Say hello to the name.

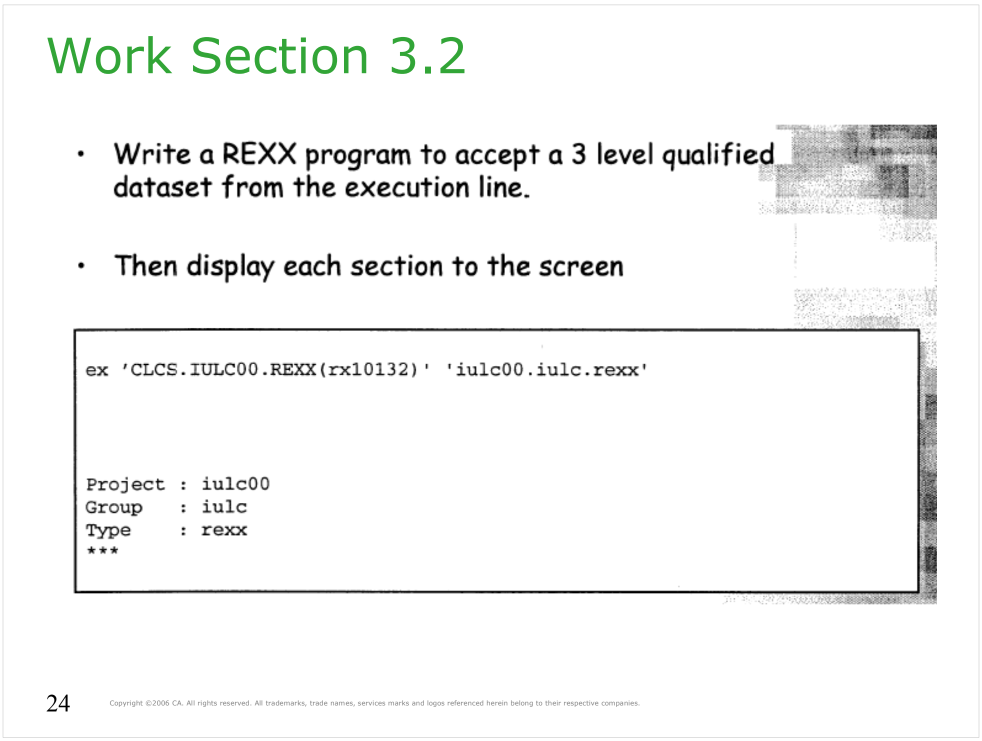
```
ex 'clcs.iulc00.rexx(rx10131)' 'bob'



Hello bob
***
```

23

Invoke the program from option 6 COMMAND



## Work Section 3.2

- Write a REXX program to accept a 3 level qualified dataset from the execution line.
- Then display each section to the screen

```
ex 'CLCS.IULC00.REXX(rx10132)' 'iulc00.iulc.rexx'



Project : iulc00
Group   : iulc
Type    : rexx
***
```

24

Invoke the program from option 6 COMMAND

## Additional Programs

- If you had proceeding spaces in work section 1.2 then re-write using PARSE to remove the spaces.