String Processing

Converting Strings of Characters To and From Lists

- An atom such as '*hello world*' can be regarded as a string of characters. Prolog has facilities to enable strings of this kind to be manipulated.
- For Example:
  - To join two strings such as '*Today is*' and ' *Tuesday*' to form '*Today is Tuesday*'.
  - To remove initial spaces, e.g. to replace ' *hello world*' by '*hello world*'.
  - To find the part of the string after (or before) a specified string, e.g. to find the name in a string such as '*My name is John Smith*'.

Converting Strings…

- Prolog does this by converting strings to equivalent lists of numbers using the **name/2** predicate and then using list processing techniques before (generally) converting the resulting lists back to strings.
- The **name/2** predicate takes two arguments. If the first is an atom and the second is an unbound variable, evaluating a **name** goal will cause the variable to be bound to a list of numbers equivalent to the string of characters that forms the atom.

Converting Strings…

- For example:

```
?- name('Prolog Example',L).
L = [80,114,111,108,111,103,32,69,120,97,109,112,108,101]
```

- Each of the 256 possible characters has an equivalent ASCII value, which is an integer from 0 to 255 inclusive.

Converting Strings…

- The table of ASCII values corresponding to the most commonly used characters is reproduced here for convenience.

9	tab	40	(	59	;	93	]
32	space	41	)	60	^	94	^
33	!	42	*	61	=	95	_
34	"	43	+	62	>	96	`
35	#	44	,	63	?	97-122	a to z
36	$	45	-	64	@	123	{
37	%	46	.	65-90	A to Z	124	|
38	&	47	/	91	[	125	}
39	'	48-57	0 to 9	92	\	126	~
		58	:				

Converting Strings…

- In the example above, 80 is the ASCII value corresponding to the character **P**, 114 corresponds to **r** etc., so the list

[80,114,111,108,111,103,32,69,120,97,109,112,108,101]

- corresponds to 'Prolog Example'.

Converting Strings…

- The **name** predicate can also be used to perform the conversion in the other direction, i.e. from a list of ASCII values to an equivalent atom.

```
?-name(A,[80,114,111,108,111,103,32,69,120,97,109,112,108,101]).
A = 'Prolog Example'
```

Converting Strings…

- Once a string has been converted to list form it can be manipulated using any of the facilities available for list processing to produce a new list or lists, which can then be converted back to strings, again using the **name** predicate.

Joining Two Strings

- The predicate **join2/3** defined below shows how to join two strings (the first two arguments) to create a third string combining the two.
- The programming technique used is a typical one in string processing: convert both strings to lists, concatenate them using **append** and finally convert back the resulting list to a string.

Joining Two Strings…

```
/* Join two strings String1 and String2 to form a new
string Newstring */
join2(String1,String2,Newstring):-
    name(String1,L1),name(String2,L2),
    append(L1,L2,Newlist),
    name(Newstring,Newlist).
```

- **?- join2('prolog',' example',S).**

  S = 'prolog example'
- **?- join2('','Prolog',S).**

  S = 'Prolog'
- **?- join2('Prolog','',S).**

  S = 'Prolog'

Joining Two Strings…

- The predicate **join3/4** defined below uses **join2** twice to join three strings together.

```
/* Join three strings using the join2 predicate */
join3(String1,String2,String3,Newstring):-
    join2(String1,String2,S),
    join2(S,String3,Newstring).
```

- **?- join3('This is',' an',' example',Newstring).**

  Newstring = 'This is an example'

Trimming a String

- This example shows how to define a predicate that will 'trim' a string, i.e. remove any white space characters (spaces, tabs etc.) at the beginning or end. This is done in four stages.

Trimming a String…

Stage 1:

- Define and test a predicate **trim/2** which takes a list of integers as its first argument and an unbound variable as its second and binds the variable to the list with any elements less than or equal to 32 at the left-hand end removed. This makes use of the techniques where the elements of a list are extracted one by one using *cons*.

```
trim([A|L],L1):-A=<32,trim(L,L1).
trim([A|L],[A|L]):-A>32.
```

- **?- trim([26,32,17,45,18,27,94,18,16,9],X).**

  X = [45,18,27,94,18,16,9]

Trimming a String…

Stage 2

- Define and test a predicate **trim2/2** which takes a list of integers as its first argument and an unbound variable as its second and binds the variable to the list with any elements less than or equal to 32 at the right-hand end removed. This uses **trim** in conjunction with the **reverse** predicate.

```
trim2(L,L1):-
reverse(L,Lrev),trim(Lrev,L2),reverse(L2,L1).
```

- **?- trim2([45,18,27,94,18,16,9],X).**
- **X = [45,18,27,94]**

Trimming a String…

Stage 3

- Define and test a predicate **trim3/2** which takes a list of integers as its first argument and an unbound variable as its second and binds the variable to the list with any elements less than or equal to 32 at the beginning and/or the end removed.
- This uses **trim** to deal with the beginning of the list and **trim2** to deal with the end.

```
trim3(L,L1):-trim(L,L2),trim2(L2,L1).
```

- **?- trim3([26,32,17,45,18,27,94,18,16,9],X).**

  X = [45,18,27,94]

Trimming a String…

Stage 4

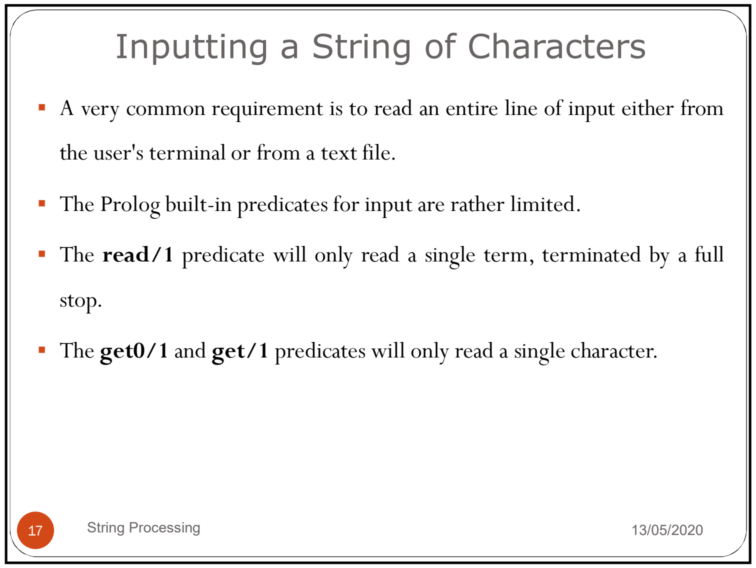
- Define and test a predicate **trims/2** which takes an atom as its first argument and an unbound variable as its second and binds the variable to the atom with any white space characters at the beginning or end removed. Now that the list processing predicates **trim**, **trim2** and **trim3** have been defined, **trims** only needs a one clause definition.

```
trims(S,Snew):-name(S,L),trim3(L,L1),name(Snew,L1).
```

- The string used as the first argument below begins with three spaces and ends with the combination 'tab, two spaces, tab'. All these are white space characters.

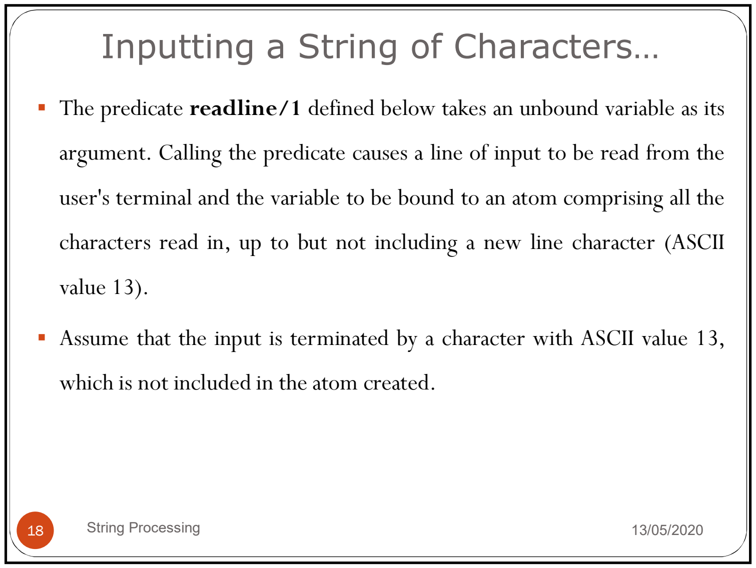
- **?- trims(' hello world ',X).**

  X = 'hello world'



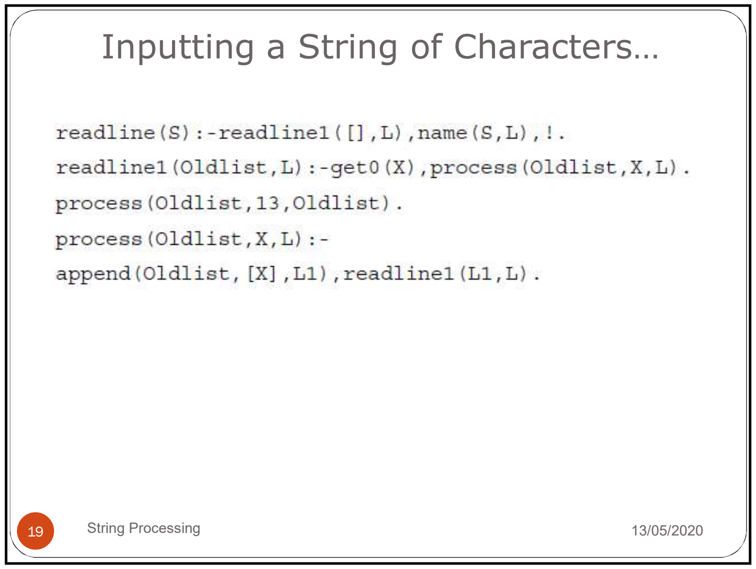
Inputting a String of Characters

- A very common requirement is to read an entire line of input either from the user's terminal or from a text file.
- The Prolog built-in predicates for input are rather limited.
- The **read/1** predicate will only read a single term, terminated by a full stop.
- The **get0/1** and **get/1** predicates will only read a single character.



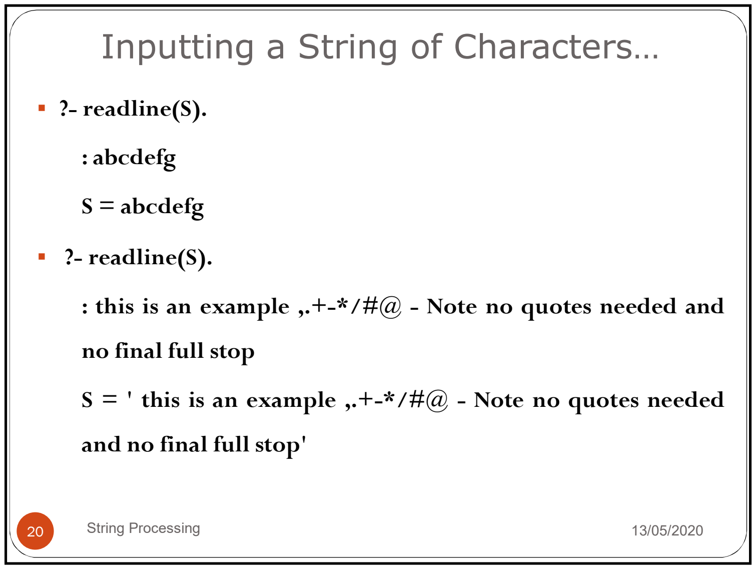
Inputting a String of Characters…

- The predicate **readline/1** defined below takes an unbound variable as its argument. Calling the predicate causes a line of input to be read from the user's terminal and the variable to be bound to an atom comprising all the characters read in, up to but not including a new line character (ASCII value 13).
- Assume that the input is terminated by a character with ASCII value 13, which is not included in the atom created.



Inputting a String of Characters…

```
readline(S):-readline1([],L),name(S,L),!.
readline1(Oldlist,L):-get0(X),process(Oldlist,X,L).
process(Oldlist,13,Oldlist).
process(Oldlist,X,L):-
append(Oldlist, [X],L1),readline1(L1,L).
```



Inputting a String of Characters…

- **?- readline(S).**

  : abcdefg

  S = abcdefg
- **?- readline(S).**

  : this is an example ,.+-*/#@ - Note no quotes needed and no final full stop

  S = ' this is an example ,.+-*/#@ - Note no quotes needed and no final full stop'

Inputting a String of Characters…

- The predicate **readlineF/2** defined below is based on **readline/1** but in an adapted form, which deals with the slightly different requirements of a string of characters read from a text file.
- It is assumed that all such lines end in two ASCII characters, with values 13 and 10, unlike lines of input from the user's terminal which end with just one character (ASCII value 13).
- Once the character with value 13 has been found, the first clause of predicate **process** ensures that the character with value 10 is read and ignored.

Inputting a String of Characters…

```
/* readline adapted for input from text files */
readlineF(File,S):-
   see(File),readline1([],L),name(S,L),!,seen.
readline1(Oldlist,L):-get0(X),process(Oldlist,X,L).
process(Oldlist,13,Oldlist):-get0(X).
/* To remove character with ASCII value 10 */
process(Oldlist,X,L):-
append(Oldlist,[X],L1),readline1(L1,L).
```

Inputting a String of Characters…

- If text file file1.txt contains the following four lines:

  This is an example of

  a text file with four lines -

  each will be terminated by two invisible characters

  with ASCII values 13 and 10
- calling **readfileF** with first argument '*file1.txt*' will cause the first line of the file to be output.
- **?- readlineF('file1.txt',S).**

  S = 'This is an example of '

Searching a String

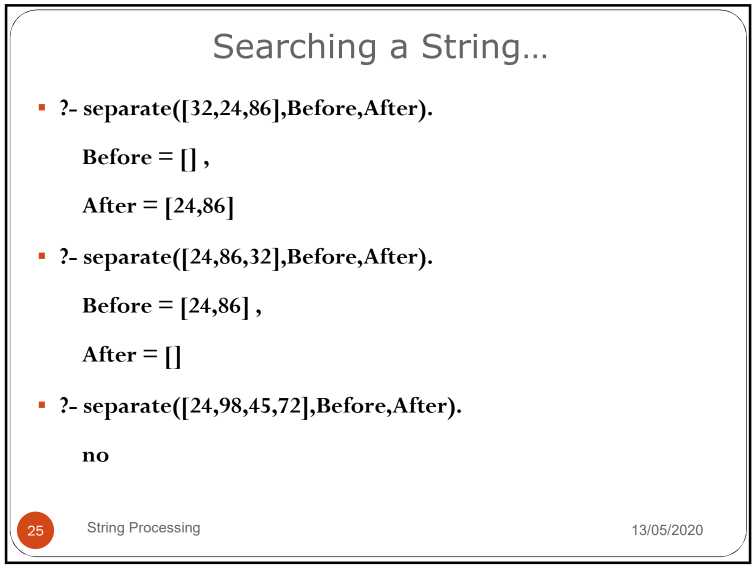
- The predicate **separate/3** defined below separates a list *L* into those elements before and after the element 32.

```
separate(L,Before,After):-
 append(Before,[32|After],L),!.
```

- It does a list with head 32 – another example of the value of using the *cons* notation to deconstruct a list.
- **?- separate([26,42,32,18,56,32,19,24],Before,After).**

  Before = [26,42] ,

  After = [18,56,32,19,24]



Searching a String…

- ?- separate([32,24,86],Before,After).

  Before = [] ,

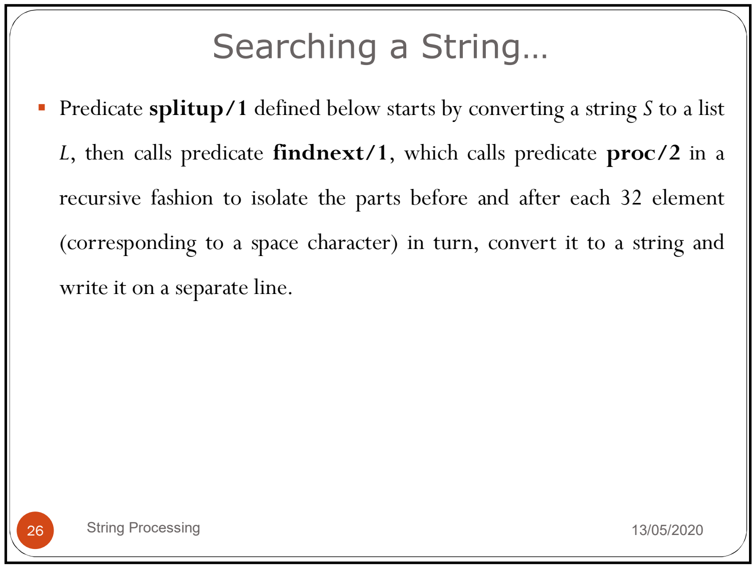
  After = [24,86]
- ?- separate([24,86,32],Before,After).

  Before = [24,86] ,

  After = []
- ?- separate([24,98,45,72],Before,After).

  no

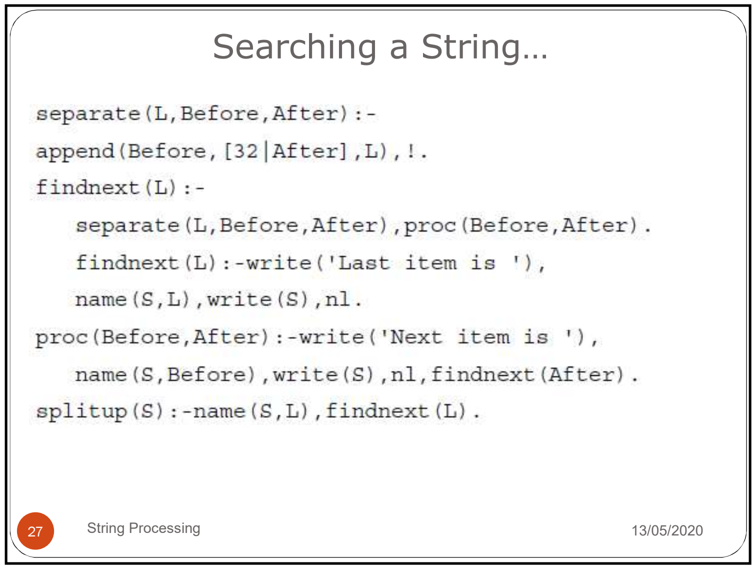
25 String Processing 13/05/2020



Searching a String…

- Predicate **splitup/1** defined below starts by converting a string *S* to a list *L*, then calls predicate **findnext/1**, which calls predicate **proc/2** in a recursive fashion to isolate the parts before and after each 32 element (corresponding to a space character) in turn, convert it to a string and write it on a separate line.

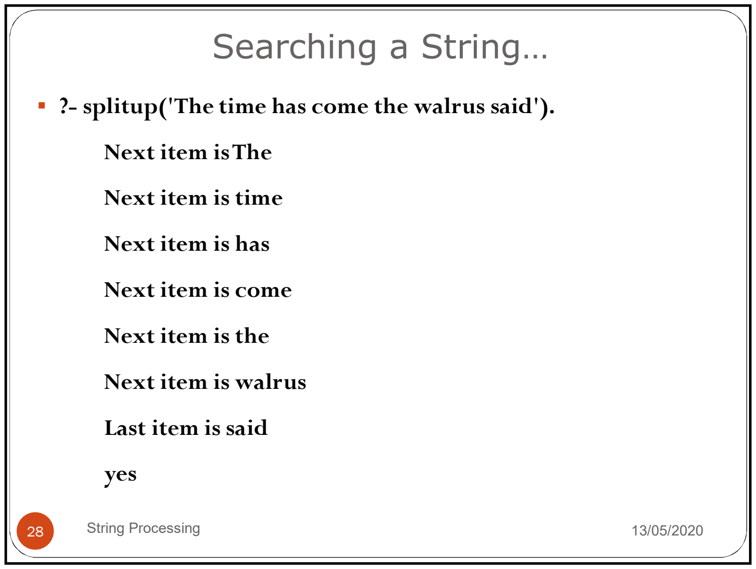
26 String Processing 13/05/2020



Searching a String…

```
separate(L,Before,After):-
append(Before,[32|After],L),!.
findnext(L):-
   separate(L,Before,After),proc(Before,After).
   findnext(L):-write('Last item is '),
   name(S,L),write(S),nl.
proc(Before,After):-write('Next item is '),
   name(S,Before),write(S),nl,findnext(After).
splitup(S):-name(S,L),findnext(L).
```

27 String Processing 13/05/2020



Searching a String…

- ?- splitup('The time has come the walrus said').

  Next item is The

  Next item is time

  Next item is has

  Next item is come

  Next item is the

  Next item is walrus

  Last item is said

  yes

28 String Processing 13/05/2020

Searching a String…

- The predicate **checkprolog/1**, which takes an unbound variable as its argument, causes a string of characters to be read from the user's terminal and the argument to be bound to either the atom *present* or the atom *absent*, depending on whether or not the input string includes the word *Prolog*.
- The main predicate defined here is **startList/2**, which uses **append** to check whether all the elements in list *L1* appear at the beginning of list *L2*.

Searching a String…

```
/* Uses predicate readline as defined previously */
startList(L1,L2):-append(L1,X,L2).
includedList(L1,[]):-!,fail.
includedList(L1,L2):-startList(L1,L2).
includedList(L1,[A|L2]):-includedList(L1,L2).
checkit(L,Plist,present):-includedList(Plist,L).
checkit(_,_,absent).
checkprolog(X):-readline(S),name(S,L),
   name('Prolog',Plist),checkit(L,Plist,X),!.
```

Searching a String…

- **?- checkprolog(X).**

  : Logic Programming with Prolog

  X = present
- **?- checkprolog(X).**

  : Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto

  X = absent

Dividing a String into Its Component Parts

- The predicate **splits/4** divides a string into the substrings to the left and right of another string called a *separator*. Its arguments correspond to the string, the separator, the left part and the right part in turn.
- For example:
  - **?- splits('In the beginning was the word','the',Left,Right).**

    Left = 'In ' ,

    Right = ' beginning was the word'
  - **?- splits('my name is John Smith','is',Left,Right).**

    Left = 'my name ' ,

    Right = ' John Smith'

Dividing a String into Its Component Parts…

- The definition of this predicate is quite complex. There are four special cases:
- **(1)** The separator appears more than once in the string. As can be seen from the first example above, the first (i.e. 'leftmost') occurrence is taken.
- **(2)** The string begins with the separator. In this case the left part should be set to the separator and the right part should be set to the remainder of the string.

Dividing a String into Its Component Parts…

- **(3)** The string ends with the separator. In this case the right part should be set to the separator and the left part should be set to the remainder of the string.
- **(4)** The separator does not appear in the string. In this case the left part should be set to the string and the right part should be set to ''.

Dividing a String into Its Component Parts…

- The full definition is given below, followed by examples showing that cases (2), (3) and (4) are dealt with correctly.

```
splits(S,Separator,Separator,R):-
   name(Separator,L1),name(S,L3),
   append(L1,L2,L3),name(R,L2),!.
splits(S,Separator,L,Separator):-
   name(Separator,L2),name(S,L3),
   append(L1,L2,L3),name(L,L1),!.
splits(S,Separator,Left,Right):-
   name(S,L3),append(Lleft,Lrest,L3),
   name(Separator,L4),append(L4,Lright,Lrest),
   name(Left,Lleft),name(Right,Lright),!.
splits(S,_,S,''):-!.
```

Dividing a String into Its Component Parts…

- **?- splits('my name is John Smith','my name is ',Left,Right).**

  Left = 'my name is ' ,

  Right = 'John Smith'

- **?- splits('my name is John Smith','John Smith',Left,Right).**

  Left = 'my name is ' ,

  Right = 'John Smith'

Dividing a String into Its Component Parts…

- **?-splits('my name is my name is John Smith','is',Left,Right).**

  Left = 'my name ' ,

  Right = ' my name is John Smith‘

- **?- splits('my name is John Smith','Bill Smith',Left, Right).**

  Left = 'my name is John Smith' ,

  Right = ''

Dividing a String into Its Component Parts…

- Predicate **remove_spaces/2** defined below uses predicate **splits/4** to removeany initial spaces from a string. The key idea is to split the string using a string containing a space, i.e. ' ', as the separator. If the left part becomes bound to ' ' it implies that case (2) above has occurred, i.e. the string must have begun with theseparator.
- In this case (first clause of **remove2**), the remainder of the string (the right part) is used and any further spaces are removed from it in the same way. If not, the final string is the same as the original string (second clause of **remove2**).

Dividing a String into Its Component Parts…

```
remove_spaces(S,S1):-
   splits(S,' ',Sleft,Sright),
   remove2(S,Sleft,Sright,S1),!.
remove2(S,' ',Sright,S1):-remove_spaces(Sright,S1).
remove2(S,_,_,S).
```

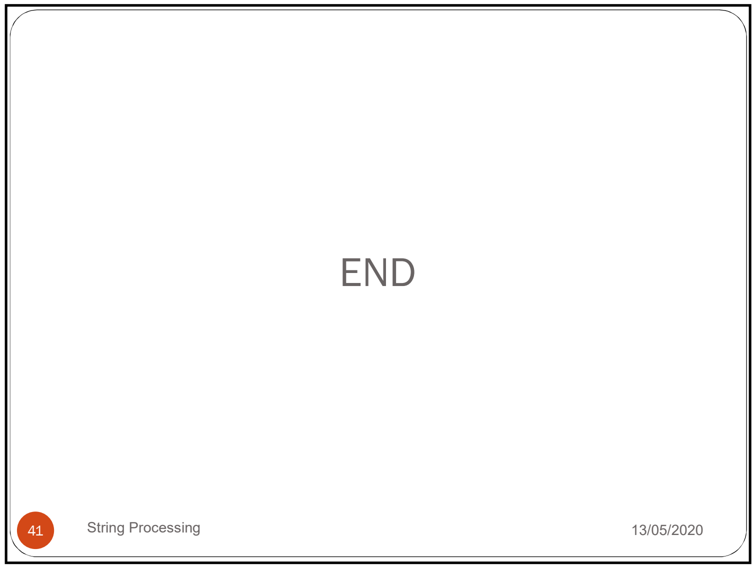
Dividing a String into Its Component Parts…

- **?- remove_spaces('hello world',X).**

  X = 'hello world'

- **?- remove_spaces(' hello world',X).**

  X = 'hello world'



END

41 String Processing 13/05/2020