2015 Survey Solutions
Washington DC, USA

Survey Solutions: String Functions

Sergiy Radyakin

sradyakin@worldbank.org

Research Department (DECRG),
The World Bank

December 20, 2016

- Strings are one of the most frequest data types used for capturing names, addresses, titles, descriptions, etc.
- From the software point of view strings are sequences of characters (type of question: text).
- Strings as constants are spelled in quotes: `"this is a string"` and can include spaces and digits and other characters.
- Similarly to how mathematical operators apply to numbers, there are functions defined that allow processing string data.
- Functions useful in processing strings may return as a result a number, a logical value, another string, or a value of another type.

- strings can be tested for equality or inequality;
- comparison can be done with an operator or a function;
- two strings are equal when they have identical content; if `city` is a string variable, then the expression

  `city=="Boston"`

  is *true* only if the value of the variable `city` is exactly *"Boston"*;
- strings are compared in dictionary order: `"A"<"B"` is *true*, and so is `"AAC"<"AB"`; note that shorter string can still be 'larger' than a longer string.
- In C# we don't use the `"<"` operator. We use a special function to compare strings.

- Two strings can be compared with the `String.Compare()` function;
- It takes 3 arguments: *str1*, *str2*, and a logical flag whether case matters or should be ignored;
- For example: `String.Compare(s,"New York",true)` compares the value in variable s with a string constant *"New York"* ignoring the case.
- The result of this function is:
  - Less than zero, if *str1* precedes *str2* in the sort order;
  - Zero, if *str1* occurs in the same position as *str2* in the sort order;
  - Greater than zero, if *str1* follows *str2* in the sort order.

- Comparing strings is important: recall that Survey Solutions distinguishes two kinds of expressions (by their purpose): validation expressions and enablement conditions, both are of logical (Boolean) type;
- it is incorrect to e.g. specify the particular string values in any of these expression fields, like so:

```
"Washington", "Jackson", "Franklin";
```

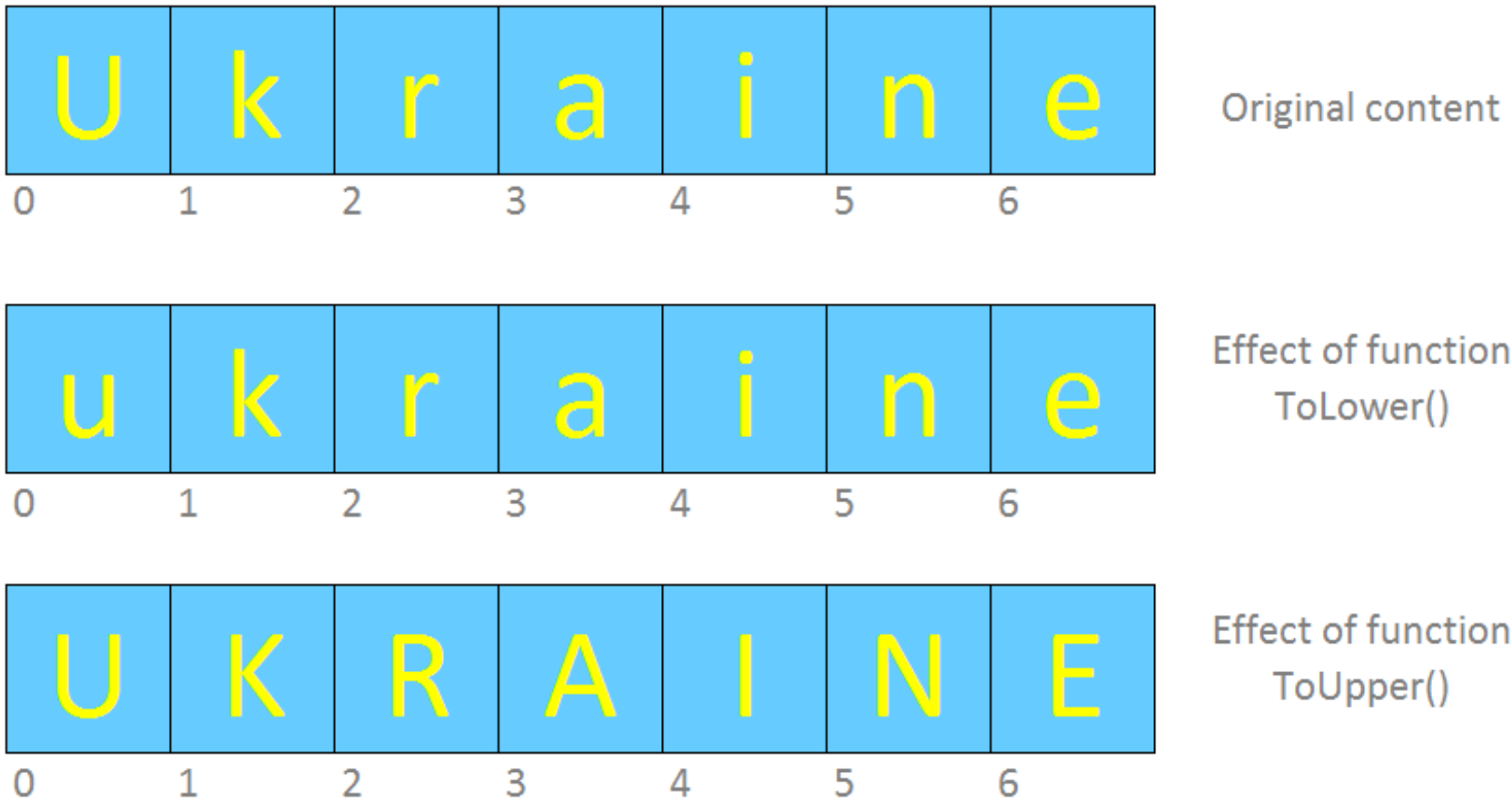
- instead a logical expression must be formed, which evaluates to *true* or *false*, such as:

```
name=="Washington" || name=="Jackson" ||
                      name=="Franklin"
```

or using an alternative notation:

```
name.InList("Washington","Jackson","Franklin")
```

- Functions *ToLower()* and *ToUpper()* can be used to convert all characters to either all lower or all upper characters:



For example:
`"Ukraine".ToLower()` evaluates to “ukraine”.
`"Ukraine".ToUpper()` evaluates to “UKRAINE”.

Use case transformations when you need to verify the content is equal to some other content and you want case insensitive comparison.

For example, the result of the following expression

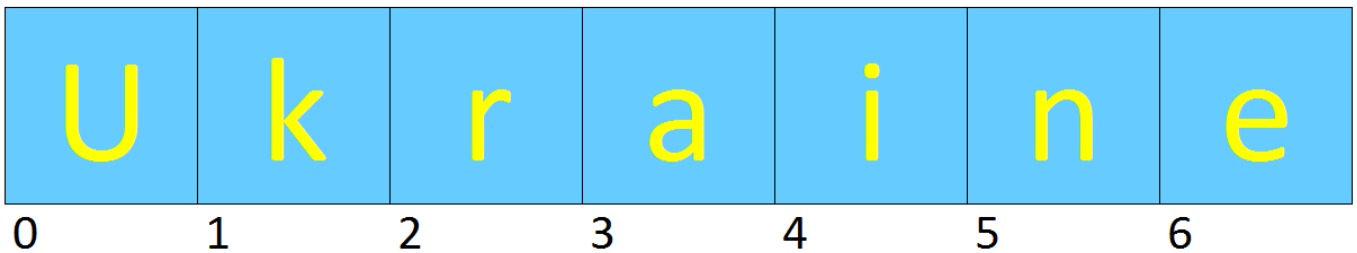
```
s.ToLower()=="ukraine"
```

will be *true* when s is *“ukraine”*, *“Ukraine”*, and *“UKRAINE”*, but is *false*, for example, for *“Ukrain”*.

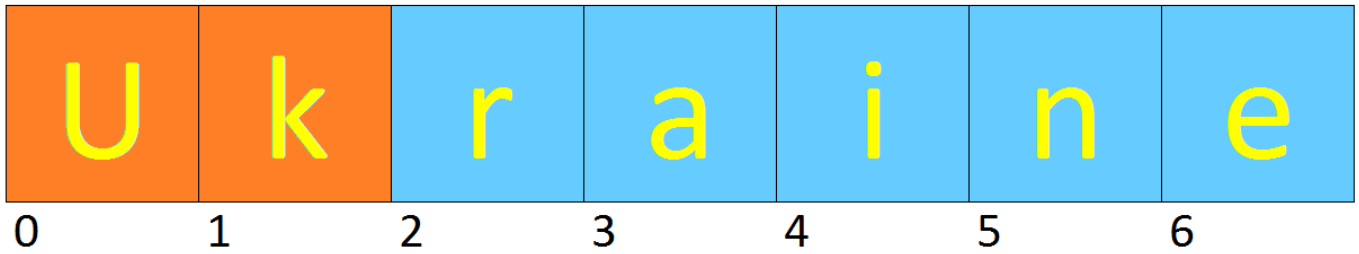
- It is generally not advisable to require interviewers to enter data using an upper or lower case;
- This requirement historically comes from the need to make certain important information (like names) more readable, and for this reason in paper questionnaires an instruction was placed to make interviewers use capital letters. With an electronic data entry system this is no longer required.
- If it is known that certain information is, for example, all upper case (like the flight numbers), the transformation can be done later, on the final exported dataset, using whatever system is being employed for cleaning up and reshaping the data;
- This speeds up data entry, and does not distract the interviewer's attention on non-significant details.

- Strings (as sequences of characters) have length. The more characters are in the string the longer it is. The function `Length` allows determining the length of a string: `"Ukraine".Length` returns 7.
- `Length` receives no arguments, and is written without parentheses (technically it is not a function, it's a property of a string).
- `Length` can be used to check if a certain string is empty (has no characters) or ensure that the content is longer than a certain minimal threshold, for example that the description of occupation is at least 20 characters or name is longer than 2 characters.
- Remember that spaces, commas, periods, and other characters are counted towards the length of the string.

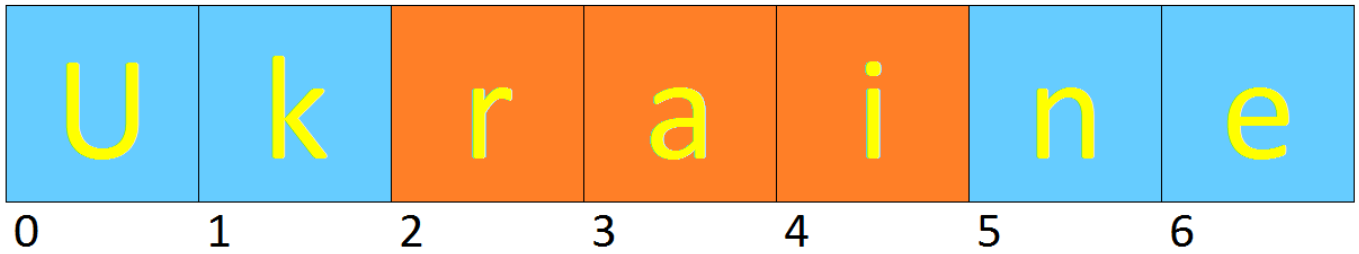
- A particular portion of the string can be extracted for further analysis.
- Recall that the string is a sequence of characters, with every character occupying one position:



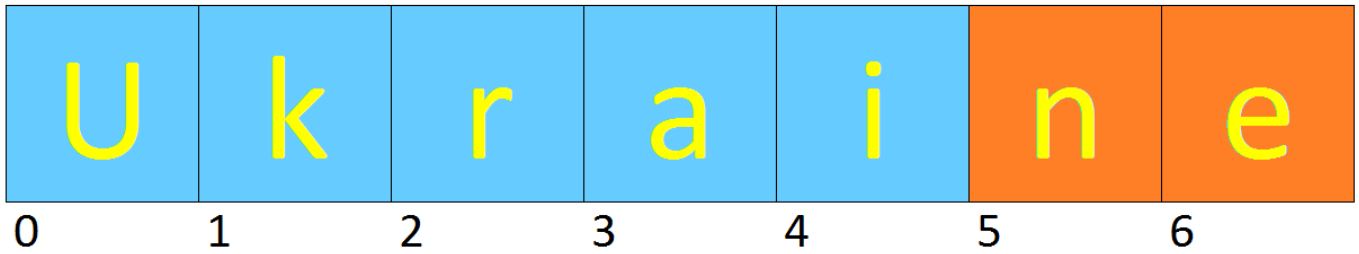
- One can instruct a substring to be extracted by specifying the proper location and number of characters.
- Important: the indexing of characters starts with 0 (the first character in the string has the index of zero).



If text question identified by variable name *country* takes a value *"Ukraine"* then the expression `country.Left(2)` evaluates to *"Uk"*.



If text question identified by variable name *country* takes a value *"Ukraine"* then the expression `country.Substring(2,3)` evaluates to *"rai"*.



If text question identified by variable name *country* takes a value *"Ukraine"* then the expression `country.Right(2)` evaluates to *"ne"*.

String concatenation is an operation of combining multiple strings into a single long string. This can be done with the use of the `Concat()` function.

For example, if variable *name* has value "George" then:

- `Concat(name,"!")=="George!"` is *true*;
- `Concat(name,"Washington")=="George Washington"` is *false*;
- `Concat(name," ","Washington")=="George Washington"` is *true*;

An operator + can be used to concatenate strings:

```
"George"+" "+"Washington"=="George Washington"
```

evaluates to *true*.

- Character "space" and some other characters constitute non-missing non-printable content; (for clarity in this slide □ is used to denote a whitespace);
- Strings *"New□York"* and *"New□York□"* are different, the second one ends in a space;
- Commonly whitespace needs to be ignored during processing; this is relatively easy to do whith a statistical package; in Survey Solutions use functions `TrimStart()`, `TrimEnd()`, and `Trim()`:
  - *TrimStart()* removes leading whitespace,
  - *TrimEnd()* removes trailing whitespace, and
  - *Trim()* removes both leading and trailing whitespace;
- If s has a value "□New□York□", then `s.TrimStart()` is *"New□York□"*, `s.TrimEnd()` is *"□New□York"* and `s.Trim()` is *"New□York"*;
- Note that the space in the middle is not removed by any of the trimming functions.

- A common task is to check whether the string starts or ends with a specific word. This can be done conveniently with the `StartsWith()` and `EndsWith()` functions.
- Suppose a text question with variable name s has a value *"Hello, World!"*.
- Then `s.StartsWith("Hello")` is *true*, but `s.StartsWith("Hi")` is *false*, and `s.StartsWith("HELLO")` is also *false* (comparisons are case-sensitive).
- Similarly, `s.EndsWith("!")` is *true*, and `s.EndsWith(".")` is *false*.

Both functions below are logical functions returning values *true* or *false*, and are preferred over comparing with the "" empty string for checking if the question was answered:

- To check whether the string is empty, use the `String.IsNullOrEmpty()` function, like so:

  ```
  String.IsNullOrEmpty(name)
  ```

  the result of the function is *true* when the question identified by the variable *name* is not answered;
- Similarly, `String.IsNullOrWhitespace(name)` will check if the question is not answered, or contains only whitespace characters;

Both functions below are logical functions returning values *true* or *false*:

- One can check for the content to be solely of Latin letters with the `IsAlphaLatin()` function, like so:

```
name.IsAlphaLatin()
```

- Similarly one can additionally permit for delimiters using a function `IsAlphaLatinOrDelim()`, like so:

```
region.IsAlphaLatinOrDelim()
```

- To check for other characters one can use the function `ConsistsOf()`, for example:

  ```
  ip.ConsistsOf("1234567890.")
  ```

  will check that the IP address as entered consists of only the numeric characters or a dot (this is a simplified verification of the IP, which will not catch the errors like `".1.2."`, or `"1..2"`, etc).
- One frequent use of this function is to restrict entry to a particular language, such as only English, or only Russian;
- Entry in any other language would result in the use of characters other then permitted, and an indication of an error to the interviewer;
- Note, that the program does not 'understand' the language, of course, so if two or more languages are using the same set of letters, the program will permit entry in any of them.

Function `IsLike()` allows checking for a particular pattern. For example, if s has value "Hello, World!" then:

- `s.IsLike("Hello, *")` is *true*;
- `s.IsLike("He??o, World!")` is *true*;
- `s.IsLike("He*o, World!")` is *true*.

Note how the use of the question mark and star symbols. A question mark (?) works like a wildcard, which means *"any one character"*, and a star (*) is a wildcard with meaning *"any character, several characters, or nothing"*.

To see whether a string variable contains a particular substring, use the function `Contains()`. If variable country has value *"United States"* then:

- `country.Contains("States")` is *true*;
- `country.Contains("United")` is *true*;
- `country.Contains("USA")` is *false*;

Note that the function returns a logical value, indicating whether the argument is present in the string or not, which is convenient for using in the validation or enablement conditions. However it does, not indicate whether the value occurs once, or multiple times or at what position.

A similar function to check whether a string variable contains a particular substring, is the function `IndexOf()`. It returns a zero-based index of first occurrance of the specified substring in the string, or -1 if not found. For example, if variable country has value *"United States"* then:

- `country.IndexOf("States")` is 7;
- `country.IndexOf("United")` is 0;
- `country.IndexOf("USA")` is -1;

Note that since the indexing starts with zero, then the logical expression one must use for checking the presence of a substring is "more or equal than zero":

```
source.IndexOf("target")>=0
```

- `IsNumber()` verifies if the content represents a number; for example:
  `"5.3".IsNumber()` evaluates to *true*;
  `"five".IsNumber()` evaluates to *false*.

- `IsIntegerNumber()` verifies if the content represents an integer number; for example:
  `"5".IsIntegerNumber()` evaluates to *true*;
  `"5.3".IsIntegerNumber()` evaluates to *false*.

- `IsMilitaryTime()` verifies if the string represents time in military notation, for example:
  `IsMilitaryTime("06:30")` evaluates to *true*;
  `IsMilitaryTime("6:30AM")` evaluates to *false*;

- `IsMilitaryTimeZ()` verifies if the string represents time in military notation with the time zone specified, for example:
  `IsMilitaryTimeZ("17:45B")` evaluates to *true*;
  `IsMilitaryTimeZ("17:65B")` evaluates to *false*;

- `IsValidEmail()` verifies if the string represents an email address, for example:

  `email.IsValidEmail()` evaluates to *true* if variable *email* contains *"alpha@beta.com"*;

  `email.IsValidEmail()` evaluates to *false* if variable *email* contains *"alphabeta.com"* (an invalid email address).

Many checks can be performed equivalently using different functions. For example, any of the expressions below can be used to check whether a string variable *flight* starts with letters *"UA"*:

- `flight.StartsWith("UA")`
- `flight.Left(2)=="UA"`
- `flight.IndexOf("UA")==0`
- `flight.IsLike("UA*")`

Combining functions provides a possibility to do more complex checks. Suppose we wanted to check if a particular string is an American Airlines flight number. Let's inspect the AA flights schedule to understand the numbering pattern:

To New York (JFK), NY, US (JFK)
187 Miles

6:05 AM	7:20 AM		**AA1165**	738	0	1h 15m
Above Flt Disc. JUN 3						
6:05 AM	7:19 AM		**AA330**	738	0	1h 14m
Above Flt Eff. JUN 4						
9:29 AM	10:50 AM	23	**AA84**	738	0	1h 21m
Above Flt Disc. MAY 6						
9:29 AM	10:50 AM		**AA236**	738	0	1h 21m
Above Flt Eff. MAY 7 thru JUN 3						
9:31 AM	10:53 AM		**AA236**	738	0	1h 22m
Above Flt Eff. JUN 4						
10:05 AM	12:05 AM+1	6	**AA1898✦**	CHG	2	14h 00m
Operated by US Airways; Above Flt Eff. JUN 6						
11:10 AM	4:10 PM	6	**AA2017✦**	319	1	5h 00m
Operated by US Airways; Above Flt Eff. MAY 9 thru MAY 30						
2:40 PM	4:04 PM	23	**AA178**	738	0	1h 24m
Above Flt Disc. MAY 6						

It appears American Airlines uses "AA" followed by 2,3, or 4-digit numeric codes for flight numbers, and the first digit may not be a "9", (which is reserved for empty flights). Whether true or not, let's pretend this *is* the definition that we want to check.

```
flight.Length.InRange(4,6) && flight.StartsWith("AA") &&
    (flight(2,flight.Length-2)).IsIntegerNumber &&
!((flight(2,flight.Length-2))).StartsWith("0") && !flight.IsLike("AA9*")
```

- checks length of value to be at least 4 characters, but no longer than 6 characters;
- checks value starts with "AA";
- checks that the remainder represents an integer number;
- excludes values like "AA00", "AA000", and "AA0000", which are not valid flight numbers and like "AA017" and "AA0084", which are improperly formatted entries;
- furthermore excludes values which contain 9 as the first digit.

This expression uses very useful *InRange()* and *InList()* functions, which are not discussed here.

See their description in the Functions Reference Guide online.

- Sometimes data is entered as non-string, but then for one reason or another it is more convenient to work with it in a string form.
- In such cases a non-string values can be converted to their string representation with a `ToString()` function.
- For example, if question with variable `birthday` was capturing date of birth, then `birthday.Value.Year` would refer to just the year of birth and correspondingly `birthday.Value.Year.ToString()` to its string representation. To verify that the person was born in the 1980s one can write `birthday.Value.Year.InRange(1980,1989)` or alternatively in the string notation `birthday.Value.Year.ToString().IsLike("198?")`

Standard C#	Survey Solutions
Compare()	Concat()
ToLower(), ToUpper()	IsAlphaLatin()
Length	IsAlphaLatinOrDelim()
Substring()	Left(), Right(), ConsistsOf()
TrimStart(), Trim(), TrimEnd()	IsLike()
StartsWith(), EndsWith()	IsNumber()
IsNullOrEmpty(), IsNullOrWhitespace()	IsIntegerNumber()
Contains(), IndexOf()	IsMilitaryTime()
ToString()	IsMilitaryTimeZ()
	IsValidEmail()

Additional information for standard C# functions is generally available from multiple internet sources. For Survey Solutions specific functions additional information is available in the *Functions Reference Guide* online.

You can find more information by following these links to resources produced by third-parties in the Internet:

- https://msdn.microsoft.com/en-us/library/ms228362.aspx
- https://msdn.microsoft.com/en-us/library/system.string_methods
- http://www.tutorialspoint.com/csharp/csharp_strings.htm
- http://www.dotnetperls.com/string

1. On the nationality question q1 in a hypothetical questionnaire the interviewers may enter values "Bhutan", "BHUTAN", "Bhutanese", "Kingdom of Bhutan", and other variations. Construct an expression that would enable the following question q2 only for Bhutan nationals.
2. A hypothetical questionnaire collects information on the universities where a degree was obtained. Interviewers are instructed to enter the university name followed by the hash sign "#" followed by the city name, where the university is located (both pieces of iformation are deemed required). Construct a validation expression for this question. Suggest an alternative questionnaire design.
3. In a migrant workers survey, respondents are asked which airport did they arrive to when they came back from the last trip. Construct a validation expression for the question capturing a 3-letter IATA airport code for your country. Discuss an alternative questionnaire design.