String in C#

Lesson Objectives

- String in C#
- StringFormat
- String Interpolation
- StringBuilder
- Regular Expression

Section 1

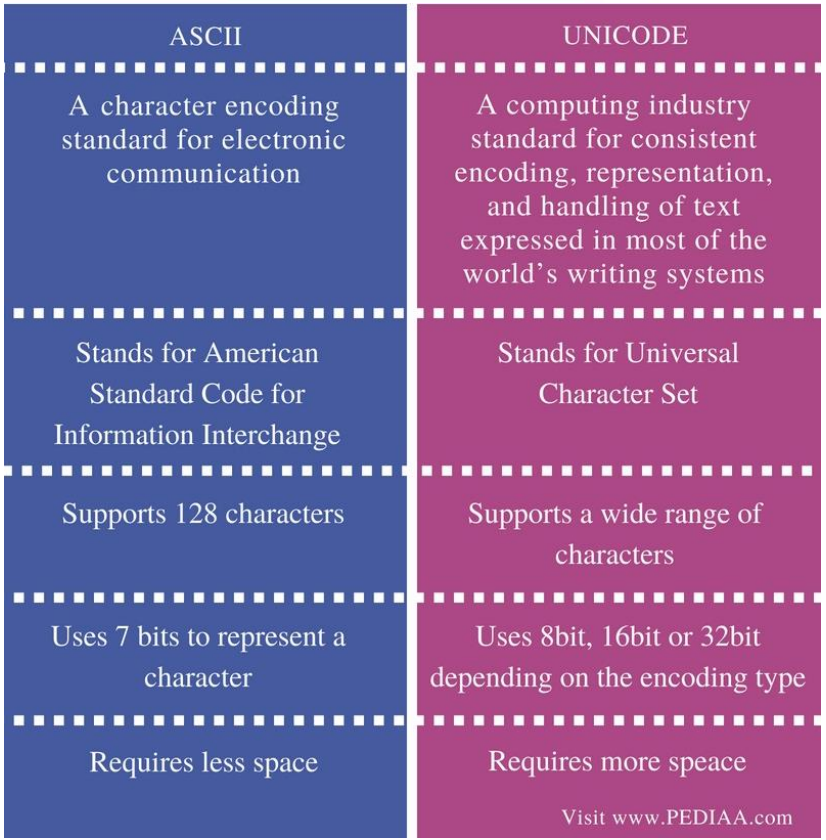
STRING IN C#

String in C#

- A string is an object of type String whose value is text.
- The text is stored as a sequential read-only collection of Char objects.
- There is no null-terminating character at the end of a C# string;
- C# string can contain any number of embedded null characters ('\0').

ASCII vs. Unicode character

- ASCII represents lowercase letters (a-z), uppercase letters (A-Z), digits (0-9) and symbols such as punctuation marks
- Unicode represents letters of English, Arabic, Greek etc., mathematical symbols, historical scripts, and emoji covering a wide range of characters than ASCII.



string vs. System.String

- In C#, the string keyword is an alias for String.
- String and string are equivalent, and you can use whichever naming convention you prefer.
- The String class provides many methods for safely creating, manipulating, and comparing strings.
- In addition, the C# language overloads some operators to simplify common string operations.

First example

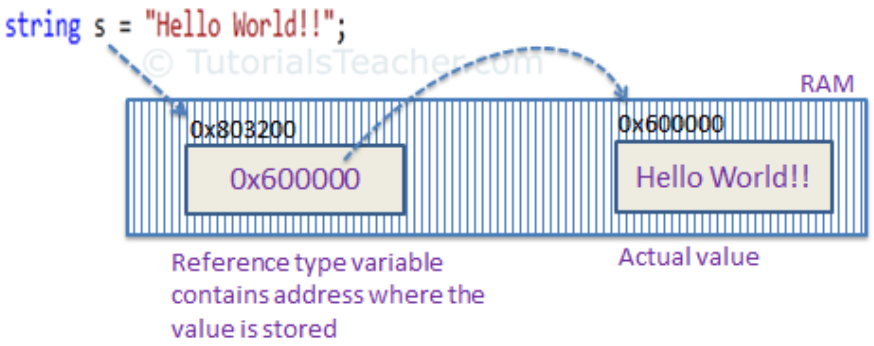
```
// Basic simple.
string greeting = "Hello World!";

// In local variables, you can use implicit typing.
var temp = "I'm still a strongly-typed System.String!";

// Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Use a const string to prevent 'message4' from
// being used to store another string value.
const string message4 = "You can't get rid of me!";
```

String operators

string firstName = "Tony";	string lastName = "Stark";	Result
firstName + lastName		"TonyStark"
firstName == lastName		false
firstName != lastName		true

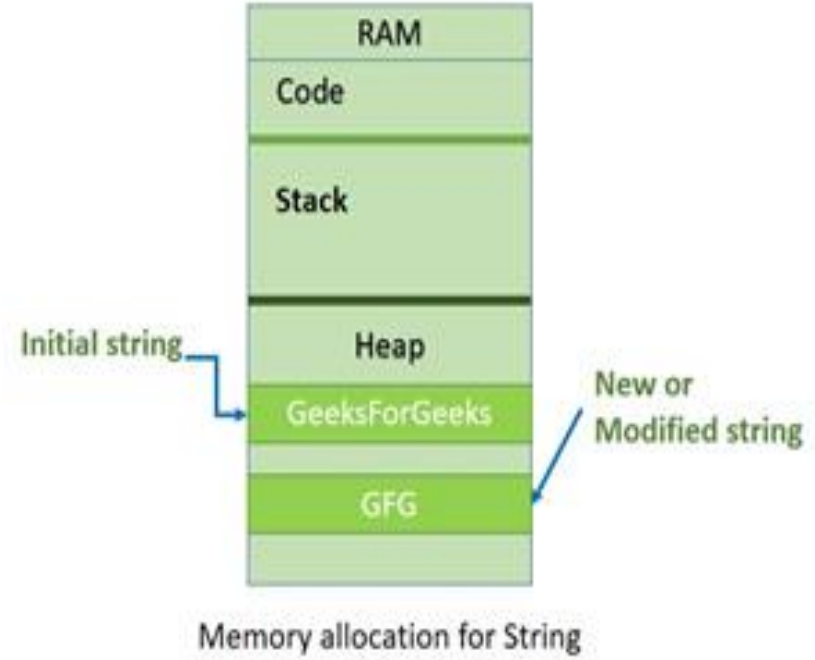
String is a Reference type

- Stored on the heap
- No default value
- Nullable
- BUT: Behaves Like Value Type



Immutability of String Objects

- String objects are immutable: they cannot be changed after they have been created.
- All of the String methods and C# operators that appear to modify a string actually return the results in a new string object.



Memory allocation for String

String Escape

Escape sequence	Character name	Unicode encoding
\'	Single quote	0x0027
\"	Double quote	0x0022
\\	Backslash	0x005C
\b	Backspace	0x0008
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009

String Escape

```
//// Use quotes inside string
string description = "We are the so-called \"Vikings\" from the north.";

//// Your local path contains backslash
string path = "D:\\Training\\Fresher Academy\\String in Csharp";

//// User \r\n to break new line
string twoLines = "No. 17, Duy Tan street\r\nCau Giay, Ha Noi";
```

Accessing Individual Characters

- Use array notation with an index value to acquire read-only access to individual characters
- It works like a array of characters

```
string s5 = "Printing backwards";

for (int i = 0; i < s5.Length; i++)
{
    System.Console.Write(s5[s5.Length - i - 1]);
}
// Output: "sdrawkcab gnitnirP"
```

Null Strings and Empty Strings

- An empty string is an instance of a System.String object that contains zero characters.
- Empty strings are used often in various programming scenarios to represent a blank text field.
- You can call methods on empty strings because they are valid System.String objects

- A null string does not refer to an instance of a System.String object and any attempt to call a method on a null string causes a NullReferenceException.
- You can use null strings in concatenation and comparison operations with other strings.

String methods

- Compare string
  - ✓ Logical comparison
  - ✓ Equal method
  - ✓ Compare method
  - ✓ IsNullOrEmpty
  - ✓ IsNullOrWhiteSpace

String methods

- Concatenate strings
  - ✓ Add operator: for simple
  - ✓ Concat method
  - ✓ Join method
  - ✓ StringBuilder

String methods

- Standardize string
  - ✓ Trim, TrimStart, TrimEnd
  - ✓ Split
  - ✓ IndexOf, IndexOfAny, LastIndexOf
  - ✓ ToLower
  - ✓ ToUpper

String methods

- Extract string
  - ✓ Contains
  - ✓ Substring
  - ✓ StartWith
  - ✓ EndWith

Practice time

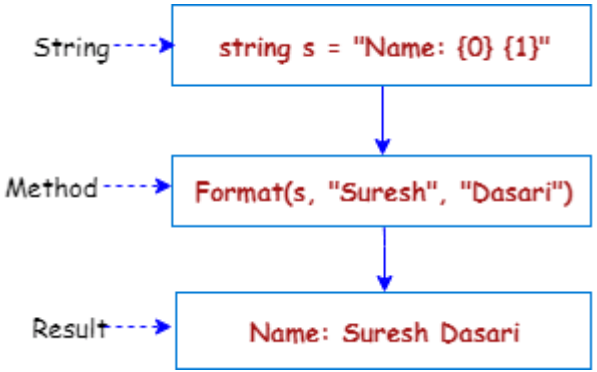
- Problem: Give a string contains some number. Use string method to extract all number and get some of them
- Example: “Peter has 3 oranges, 4 apples and 1.2 kg strawberry” should return 8.2
- Create a method to resolve problem
  - ✓ Input: string
  - ✓ Output: decimal

Section 2

STRING FORMAT

String.Format

- Converts the value of objects to strings based on the formats specified and inserts them into another string.
- Use String.Format if you need to insert the value of an object, variable, or expression into another string.



String.Format

- Benefit:
  - ✓ Clear & Clean
  - ✓ Reuseable
  - ✓ Avoid bug
- Example:
  - ✓ Give:
    - string firstName = "Tony";
    - string lastName = "Stark";
    - int age = 53;
    - int suite = 85;
  - ✓ Create string: “Tony Stark (53 years old) has made and used around 85 different suites.”

Control formatting

- You can follow the index in a format item with a format string to control how an object is formatted.

```
string s = String.Format("It is now {0:d} at {0:t}", DateTime.Now);
Console.WriteLine(s);
// Output similar to: 'It is now 4/10/2015 at 10:04 AM'
```

Standard numeric format strings

Format specifier	Name	Description	Examples
"C" or "c"	Currency	Result: A currency value.	123.456 ("C", en-US) -> $123.46
"D" or "d"	Decimal	Result: Integer digits with optional negative sign.	1234 ("D") -> 1234
"E" or "e"	Exponential (scientific)	Result: Exponential notation.	1052.0329112756 ("E", en-US) -> 1.052033E+003
"F" or "f"	Fixed-point	Result: Integral and decimal digits with optional negative sign.	1234.567 ("F", en-US) -> 1234.57
"G" or "g"	General	Result: The more compact of either fixed-point or scientific notation.	-123.456 ("G", en-US) -> -123.456
"N" or "n"	Number	Result: Integral and decimal digits, group separators, and a decimal separator with optional negative sign.	1234.567 ("N", en-US) -> 1,234.57
"P" or "p"	Percent	Result: Number multiplied by 100 and displayed with a percent symbol.	1 ("P", en-US) -> 100.00 %
"R" or "r"	Round-trip	Result: A string that can round-trip to an identical number.	123456789.12345678 ("R") -> 123456789.12345678
"X" or "x"	Hexadecimal	Result: A hexadecimal string.	255 ("X") -> FF

Custom numeric format strings

Format specifier	Name	Description	Examples
"0"	Zero placeholder	Replaces the zero with the corresponding digit if one is present; otherwise, zero appears in the result string.	1234.5678 ("00000") -> 01235
"#"	Digit placeholder	Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string.	1234.5678 ("#####") -> 1235
"."	Decimal point	Determines the location of the decimal separator in the result string.	0.45678 ("0.00", en-US) -> 0.46
","	Group separator and number scaling	Serves as both a group separator and a number scaling specifier. As a group separator, it inserts a localized group separator character between each group. As a number scaling specifier, it divides a number by 1000 for each comma specified.	Group separator specifier:
"%"	Percentage placeholder	Multiplies a number by 100 and inserts a localized percentage symbol in the result string.	0.3697 ("%#0.00", en-US) -> %36.97
"‰"	Per mille placeholder	Multiplies a number by 1000 and inserts a localized per mille symbol in the result string.	0.03697 ("#0.00‰", en-US) -> 36.97‰

Standard date and time format strings

Format specifier	Description	Examples
"d"	Short date pattern.	2009-06-15T13:45:30 -> 6/15/2009 (en-US)
"D"	Long date pattern.	2009-06-15T13:45:30 -> Monday, June 15, 2009 (en-US)
"f"	Full date/time pattern (short time).	2009-06-15T13:45:30 -> Monday, June 15, 2009 1:45 PM (en-US)
"F"	Full date/time pattern (long time).	2009-06-15T13:45:30 -> Monday, June 15, 2009 1:45:30 PM (en-US)
"g"	General date/time pattern (short time).	2009-06-15T13:45:30 -> 6/15/2009 1:45 PM (en-US)
"G"	General date/time pattern (long time).	2009-06-15T13:45:30 -> 6/15/2009 1:45:30 PM (en-US)
"M", "m"	Month/day pattern.	2009-06-15T13:45:30 -> June 15 (en-US)
"Y", "y"	Year month pattern.	2009-06-15T13:45:30 -> June 2009 (en-US)

Custom date and time format strings

- Rules:
  - ✓ Use character for value. Example: y for year, d for day,
  - ✓ M for month and m for minute
  - ✓ H for 24-hour and h for 12-hour (following by tt mean AM/PM)
  - ✓ Number of characters is length of value

Practice time

- Use string format with control formatting to print data as bellow:

```
// The example displays the following output:
//    City            Year  Population    Year  Population    Change (%)
//
//    Los Angeles     1940   1,504,277    1950   1,970,358        31.0 %
//    New York        1940   7,454,995    1950   7,891,957         5.9 %
//    Chicago         1940   3,396,808    1950   3,620,962         6.6 %
//    Detroit         1940   1,623,452    1950   1,849,568        13.9 %
```

String Interpolation

- Use string interpolation to **format** and include **expression results** in a result string.
- To identify a string literal as an interpolated string, prepend it with the $ symbol. You can embed any valid C# expression that returns a value in an interpolated string.

```
var title = $"On {date:dddd, MMMM dd, yyyy} " +
            $"Leonhard Euler introduced the letter e to denote " +
            $"{Math.E:F5} in a letter to Christian Goldbach.";
```

String Interpolation

- To include a brace, "{" or "}", in a result string, use two braces, "{{" or "}}“
- As the colon (":") has special meaning in an item with an interpolation expression, in order to use a conditional operator in an expression, enclose it in parentheses,

```
var quantity = 2;
string order = $"You bough: {quantity} {(quantity > 1 ? "items" : "item")}";
```

Practice time

- Use String Interpolation to update your code above

Section 3

STRINGBUILDER

StringBuilder

- String is immutable reference type
- StringBuilder represents a mutable string of characters.
  - ✓ A StringBuilder object maintains a buffer to accommodate expansions to the string.
  - ✓ New data is appended to the buffer if room is available; otherwise, a new, larger buffer is allocated, data from the original buffer is copied to the new buffer, and the new data is then appended to the new buffer.

String and StringBuilder in practice

Use String when:

- The number of changes that your app will make to a string is small.
- You are performing a fixed number of concatenation operations, particularly with string literals.
- You have to perform extensive search operations while you are building your string

Use StringBuilder when:

- You expect your app to make an unknown number of changes to a string at design time.
- You expect your app to make a significant number of changes to a string

Example

```
public static void Main()
{
   StringBuilder sb = new StringBuilder();
   ShowSBInfo(sb);
   sb.Append("This is a sentence.");
   ShowSBInfo(sb);
   for (int ctr = 0; ctr <= 10; ctr++) {
      sb.Append("This is an additional sentence.");
      ShowSBInfo(sb);
   }
}

private static void ShowSBInfo(StringBuilder sb)
{
   foreach (var prop in sb.GetType().GetProperties()) {
      if (prop.GetIndexParameters().Length == 0)
         Console.Write("{0}: {1:N0}    ", prop.Name, prop.GetValue(sb));
   }
   Console.WriteLine();
}
```

Memory allocation

- The default capacity of a StringBuilder object is 16 characters, and its default maximum capacity is Int32.MaxValue (~2 billions)
- Whenever the existing capacity is inadequate, additional memory is allocated and the capacity of a StringBuilder object **doubles up** to the value defined by the MaxCapacity property.

StringBuilder method

- **Append**
  - ✓ appends the string representation of a specified value (any data type)
- **AppendLine**
  - ✓ appends a copy of the specified string followed by the default line terminator to the end
- **AppendFormat**
  - ✓ appends the string returned by processing a composite format string

StringBuilder method

- **Insert**
  - ✓ Inserts the string representation of a value at the specified character position.
- **Replace**
  - ✓ Replaces all occurrences of a specified value (character or string) in this instance with another specified value.
- **Remove**
  - ✓ Removes the specified range of characters from this instance.
- **ToString**
  - ✓ Converts the value to a String

Practice time

- Use StringBuilder with appropriate method to build email
- All placeholders are in strings

Subject: Meeting Request – GEAR UP Alumni Leadership Academy

Dear (SCHEDULER NAME):

I hope this e-mail finds you well. As a GEAR UP (Gaining Early Awareness and Readiness for Undergraduate Programs) alum, I am writing to request a meeting with (MEMBER OF CONGRESS). I will be in Washington, DC, Jun 22-29 attending the GEAR UP Alumni Leadership Academy (GUALA), hosted by the National Council for Community and Education Partnerships (NCCEP).

I would very much like to visit with my Member of Congress to share my GEAR UP story and the many successes of (GEAR UP SITE NAME). **I am available to meet on Wednesday afternoon, June 26, 2013, between 1:00 PM and 4:30 PM.**

Hopefully (MEMBER OF CONGRESS NAME) or somebody from (HIS/HER) staff is available to meet with me. Please let me know where and what time works best. Thank you.

We look forward to hearing back from you soon!

Best,

(FULL NAME)
GEAR UP Alumni Leadership Academy
(GEAR UP SITE NAME)
(YOUR TELEPHONE NUMBER)
(YOUR FULL ADDRESS)

Section 4

REGULAR EXPRESSIONS

Regular Expression

- A **regular expression (aka: regex)** is a pattern that could be matched against an input text.
- The Regex class represents the .NET Framework's regular expression engine.
- It can be used to quickly parse large amounts of text to find specific character patterns; to extract, edit, replace, or delete text substrings; and to add the extracted strings to a collection to generate a report.

Regular Expression

```
// Define a regular expression for repeated words.
Regex rx = new Regex(@"\b(?<word>\w+)\s+(\k<word>)\b",
  RegexOptions.Compiled | RegexOptions.IgnoreCase);

// Define a test string.
string text = "The the quick brown fox  fox jumps over the lazy dog dog.";

// Find matches.
MatchCollection matches = rx.Matches(text);
```

Regular Expression

Pattern	Description
`^`	Start at the beginning of the string.
`\s*`	Match zero or more white-space characters.
`[\+-]?`	Match zero or one occurrence of either the positive sign or the negative sign.
`\s?`	Match zero or one white-space character.
`\$?`	Match zero or one occurrence of the dollar sign.
`\s?`	Match zero or one white-space character.
`\d*`	Match zero or more decimal digits.
`\.?`	Match zero or one decimal point symbol.
`\d{2}?`	Match two decimal digits zero or one time.
`(\d*\.?\d{2}?){1}`	Match the pattern of integral and fractional digits separated by a decimal point symbol at least one time.
`$`	Match the end of the string.

Main method

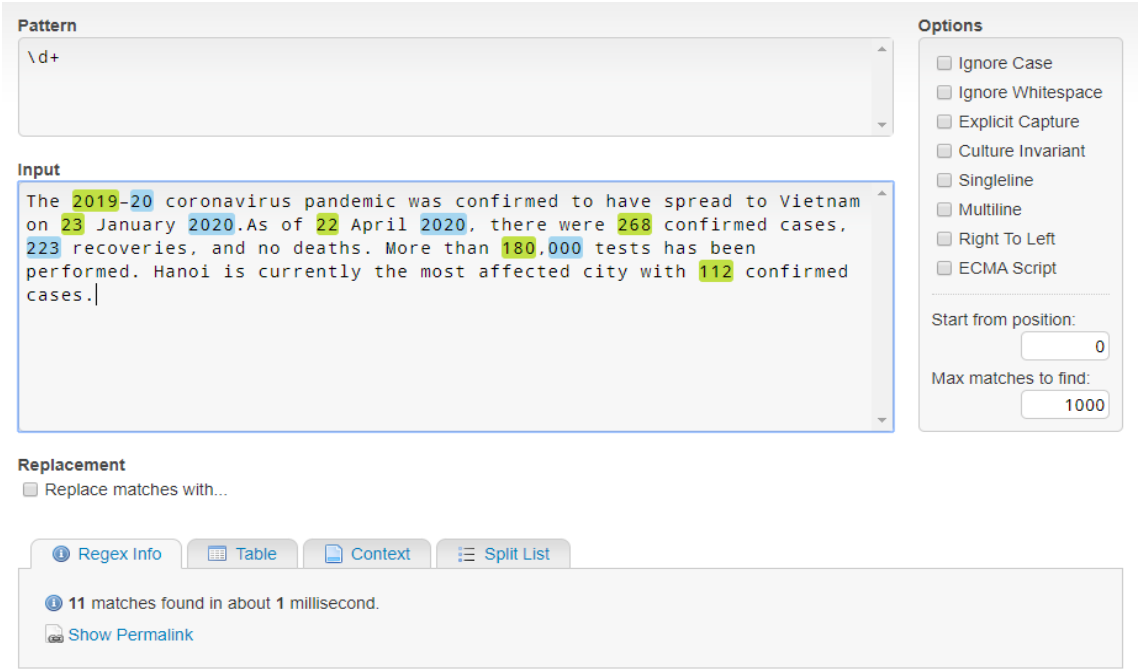
- **IsMatch**
  - ✓ Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.
- **Match**
  - ✓ Searches the specified input string for the first occurrence of the regular expression specified.

Main method

- **Replace**
  - ✓ In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.
- **Split**
  - ✓ Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified

References

- http://regexstorm.net/tester



Lesson Summary

- String in C#
- StringFormat
- String Interpolation
- StringBuilder
- Regular Expression

Thank you