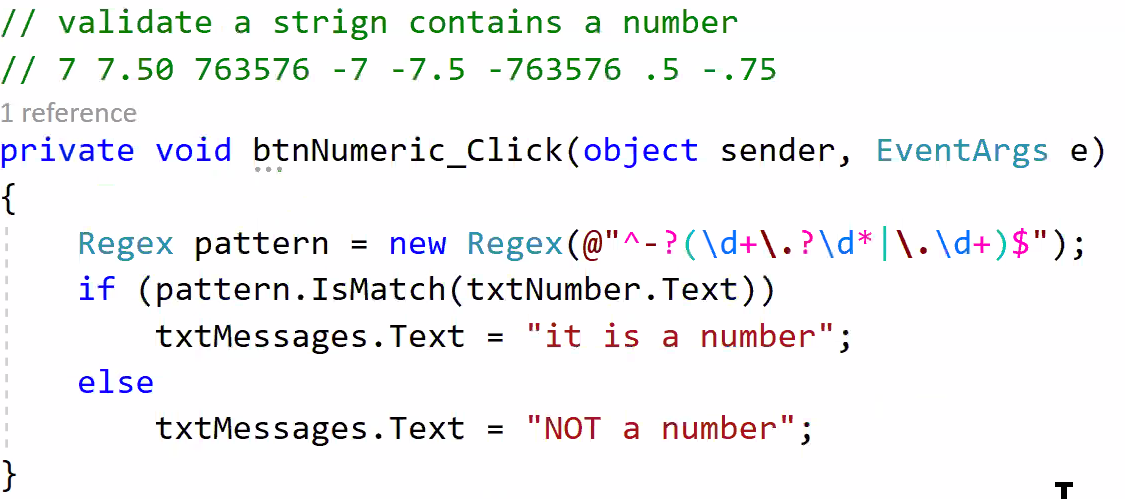
# Mid-Term Review

1. ~~Write a form that uses a regular expression to validate that the string provided contains a number. A number is defined as:~~
   1. ~~It must have at least one digit~~
   2. ~~If it has a minus sign it must be the first character~~
   3. ~~A decimal-place is optional but, if provided, there can only be one and it can be anywhere in the number~~

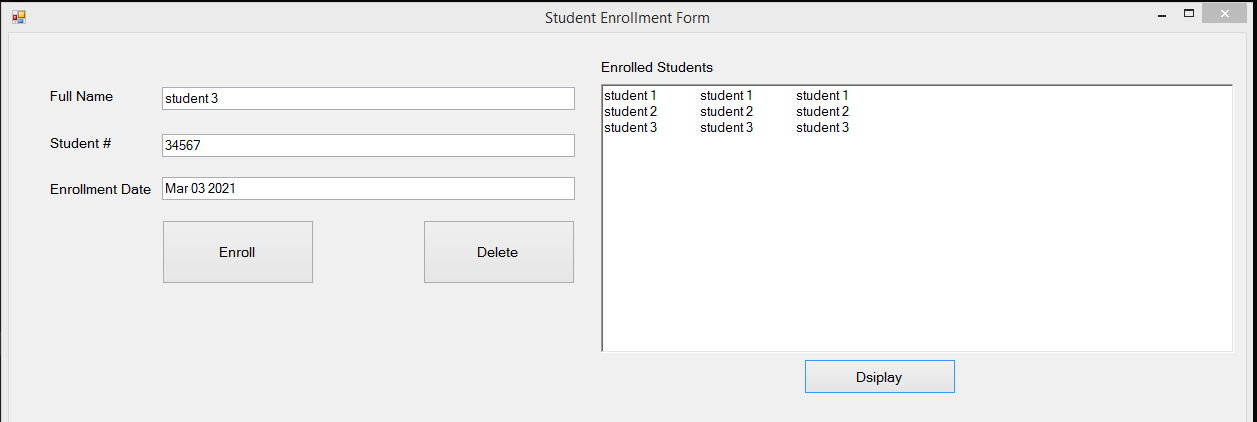


1. ~~Write a program to validate if a given number is a prime number. A prime number is a positive integer greater than 1 that is not the product of two smaller numbers … (can only be evenly divided by itself and 1).~~ 
   1. ~~The input must be a number greater than 2 digits, less than 6 digits, and not negative. Validation is required.~~
   2. ~~For example, 127 and 99989 are prime numbers and 1491 is not … while 049, 123693 and 1,033 are not valid inputs, whether prime or not.~~
2. Write a form that stores information about truck weights in an array or a collection of arrays.
   1. Fields: recordId, truckId, dateMeasured, grossWeight, tareWeight
   2. Edits:
      1. recordId is an integer and must be unique
      2. truckId is an integer greater than zero
      3. dateMeasured cannot be in the future
      4. grossWeight & tareWeight can have a decimal, must be zero or more
      5. tareWeight is not more than grossWeight
   3. Your form be able to store 5 truck records and permit updating the grossWeight of the vehicle with a given truckId
   4. Display the records in a rich text box, one line per record, separating the fields.
3. ~~Write a regular expression to validate an Ontario health card number, which looks like this:~~

~~xxxx-xxx-xxx-AA~~

~~(“AA” can be any letter pair, but must be capitals)~~

1. ~~Circle the strings that match the regular expression @”\s[A-Z]\w\*(\s|$)”~~
   1. ~~My dog’s a pug named Tazz, short for Tasmanian devil.~~
   2. ~~Mary, Mary, quite contrary … how does your garden grow?~~
   3. ~~That’s not an eh, dear, that’s an A~~
   4. ~~I got an A?~~
   5. ~~you need an M10 wrench~~
   6. ~~Is that like a 10MM wrench?~~



C# Regular Expression – Cheat Sheet

|  |  |
| --- | --- |
| **Character** | **Description** |
| **\** | Marks the next character as either a special character or escapes a literal. For example, "n" matches the character "n". "\n" matches a newline character. The sequence "\\" matches "\" and "\(" matches "(".  Note: double quotes may be escaped by doubling them: "<a href=""...>" |
| **^** | Depending on whether the MultiLine option is set, matches the position before the first character in a line, or the first character in the string. |
| **$** | Depending on whether the MultiLine option is set, matches the position after the last character in a line, or the last character in the string. |
| **\*** | Matches the preceding character zero or more times. For example, "zo\*" matches either "z" or "zoo". |
| **+** | Matches the preceding character one or more times. For example, "zo+" matches "zoo" but not "z". |
| **?** | Matches the preceding character zero or one time. For example, "a?ve?" matches the "ve" in "never". |
| **.** | Matches any single character except a newline character. |
| **(pattern)** | Matches *pattern* and remembers the match. The matched substring can be retrieved from the resulting **Matches** collection, using Item **[0]...[n]**. To match parentheses characters ( ), use "\(" or "\)". |
| **(?<name>pattern)** | Matches *pattern* and gives the match a name. |
| **(?:pattern)** | A non-capturing group |
| **(?=...)** | A positive lookahead |
| **(?!...)** | A negative lookahead |
| **(?<=...)** | A positive lookbehind . |
| **(?<!...)** | A negative lookbehind . |
| **x|y** | Matches either *x* or *y*. For example, "z|wood" matches "z" or "wood". "(z|w)oo" matches "zoo" or "wood". |
| **{*n*}** | *n* is a non-negative integer. Matches exactly *n* times. For example, "o{2}" does not match the "o" in "Bob," but matches the first two o's in "foooood". |
| **{*n*,}** | *n* is a non-negative integer. Matches at least *n* times. For example, "o{2,}" does not match the "o" in "Bob" and matches all the o's in "foooood." "o{1,}" is equivalent to "o+". "o{0,}" is equivalent to "o\*". |
| **{***n***,***m***}** | *m* and *n* are non-negative integers. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood." "o{0,1}" is equivalent to "o?". |
| **[***xyz***]** | A character set. Matches any one of the enclosed characters. For example, "[abc]" matches the "a" in "plain". |
| **[^***xyz***]** | A negative character set. Matches any character not enclosed. For example, "[^abc]" matches the "p" in "plain". |
| **[***a-z***]** | A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range "a" through "z". |
| **[^***m-z***]** | A negative range characters. Matches any character not in the specified range. For example, "[m-z]" matches any character not in the range "m" through "z". |
| **\b** | Matches a word boundary, that is, the position between a word and a space. For example, "er\b" matches the "er" in "never" but not the "er" in "verb". |
| **\B** | Matches a non-word boundary. "ea\*r\B" matches the "ear" in "never early". |
| **\d** | Matches a digit character. Equivalent to [0-9]. |
| **\D** | Matches a non-digit character. Equivalent to [^0-9]. |
| **\f** | Matches a form-feed character. |
| **\k** | A back-reference to a named group. |
| **\n** | Matches a newline character. |
| **\r** | Matches a carriage return character. |
| **\s** | Matches any white space including space, tab, form-feed, etc. Equivalent to "[ \f\n\r\t\v]". |
| **\S** | Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]". |
| **\t** | Matches a tab character. |
| **\v** | Matches a vertical tab character. |
| **\w** | Matches any word character including underscore. Equivalent to "[A-Za-z0-9\_]". |
| **\W** | Matches any non-word character. Equivalent to "[^A-Za-z0-9\_]". |
| **\num** | Matches *num*, where *num* is a positive integer. A reference back to remembered matches. For example, "(.)\1" matches two consecutive identical characters. |
| **\*n*** | Matches *n*, where *n* is an octal escape value. Octal escape values must be 1, 2, or 3 digits long. For example, "\11" and "\011" both match a tab character. "\0011" is the equivalent of "\001" & "1". Octal escape values must not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions. |
| **\x*n*** | Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, "\x41" matches "A". "\x041" is equivalent to "\x04" & "1". Allows ASCII codes to be used in regular expressions. |
| **\u*n*** | Matches a Unicode character expressed in hexadecimal notation with exactly four numeric digits. "\u0200" matches a space character. |
| **\A** | Matches the position before the first character in a string. Not affected by the MultiLine setting |
| **\Z** | Matches the position after the last character of a string. Not affected by the MultiLine setting. |
| **\G** | Specifies that the matches must be consecutive, without any intervening non-matching characters. |