1. Назовите классы, которые используются для создания меню. Перечислите свойства и методы.

Для создания меню в Windows Forms применяется элемент MenuStrip. Данный класс унаследован от ToolStrip и поэтому наследует его функциональность.

Наиболее важные свойства компонента MenuStrip:

Dock: прикрепляет меню к одной из сторон формы

LayoutStyle: задает ориентацию панели меню на форме. Может также, как и с ToolStrip, принимать следующие значения

HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина меню превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами

StackWithOverflow: элементы располагаются автоматически с переполнением

VerticalStackWithOverflow: элементы располагаются вертикально с переполнением

Flow: элементы размещаются автоматически, но без переполнения - если длина панели меню меньше длины контейнера, то выходящие за границы элементы переносятся

Table: элементы позиционируются в виде таблицы

ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню

Stretch: позволяет растянуть панель по всей длине контейнера

TextDirection: задает направление текста в пунктах меню

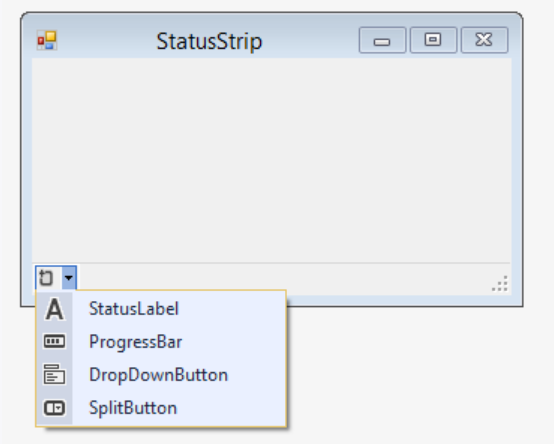
1. Что может содержать строка состояния? Какие есть методы управления строкой состояния?

StatusStrip представляет строку состояния, во многом аналогичную панели инструментов ToolStrip. Строка состояния предназначена для отображения текущей информации о состоянии работы приложения.

При добавлении на форму StatusStrip автоматически размещается в нижней части окна приложения (как и в большинстве приложений). Однако при необходимости мы сможем его иначе позиционировать, управляя свойством Dock, которое может принимать следующие значения:

* Bottom: размещение внизу (значение по умолчанию)
* Top: прикрепляет статусную строку к верхней части формы
* Fill: растягивает на всю форму
* Left: размещение в левой части формы
* Right: размещение в правой части формы
* None: произвольное положение

StatusStrip может содержать различные элементы. В режиме дизайнера мы можем добавить следующие типы элементов:



1. Что такое регулярные выражения? Где и как их можно использовать?

Регулярные выражения представляют эффективный и гибкий метод по обработке больших текстов, позволяя в то же время существенно уменьшить объемы кода по сравнению с использованием стандартных операций со строками.

Основная функциональность регулярных выражений в .NET сосредоточена в пространстве имен System.Text.RegularExpressions. А центральным классом при работе с регулярными выражениями является класс Regex. Например, у нас есть некоторый текст и нам надо найти в нем все словоформы какого-нибудь слова. С классом Regex это сделать очень просто:



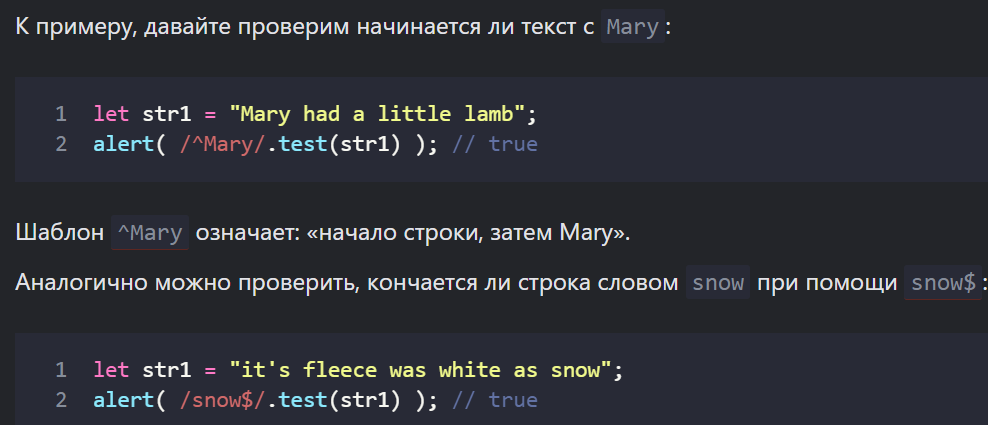
Регуля́рные выраже́ния (англ. regular expressions) — **формальный язык, используемый в компьютерных программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте**, основанный на использовании метасимволов.

Использовать можно для поиска подстрок, которые подходят по критериям, для валидации данных.

1. Что такое привязки (якоря) в RegEx? Приведите примеры.

У символов каретки ^ и доллара $ есть специальные значения в регулярных выражениях. Они называются «якоря» (anchors).

Каретка ^ означает совпадение с началом текста, а доллар $ – с концом.



1. Зачем используют конструкции группирования? Приведите примеры.

В регулярных выражениях конструкции группирования используются для определения порядка применения операторов и для создания подвыражений, которые можно использовать для поиска и замены текста.

Примеры использования конструкций группирования в регулярных выражениях:

1. Группировка операторов: вы можете использовать скобки для создания группировки операторов и управления порядком их применения. Например:
2. 

В этом примере оператор | группируется с a и b внутри скобок, а затем звездочка \* применяется ко всей группе. Результатом будет совпадение с любым количеством повторений a или b, за которым следует c.

1. Создание подвыражений: вы можете использовать скобки для создания подвыражений, которые можно использовать для поиска и замены текста. Например:



В этом примере скобки создают три подвыражения, каждое из которых соответствует группе цифр в формате XXX-XX-XXXX. Вы можете использовать эти подвыражения для поиска или замены текста в соответствии с вашими потребностями.

Конструкции группирования в регулярных выражениях являются мощным инструментом для работы с текстом. Они позволяют создавать более сложные выражения и улучшать точность поиска и замены текста.

1. Что такое квантор или множители? Приведите примеры.

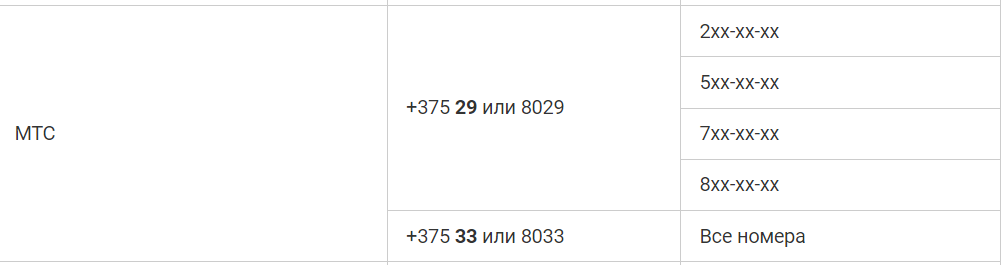
В регулярных выражениях кванторы (также называемые множителями) определяют количество раз, которое должно повторяться предыдущее символьное выражение. Они позволяют упростить и ускорить написание выражений для поиска или замены текста.

Некоторые примеры кванторов в регулярных выражениях:

1. Звездочка (\*): означает, что предыдущее символьное выражение может повторяться ноль или более раз. Например, выражение **a\*b** будет соответствовать строкам, содержащим ноль или более символов **a**, за которыми следует символ **b**.
2. Плюсик (+): означает, что предыдущее символьное выражение должно повторяться один или более раз. Например, выражение **a+b** будет соответствовать строкам, содержащим один или более символов **a**, за которыми следует символ **b**.
3. Вопросительный знак (?): означает, что предыдущее символьное выражение может быть или не быть в строке. Например, выражение **colou?r** будет соответствовать строкам, содержащим слово "color" или "colour".
4. {n}: означает, что предыдущее символьное выражение должно повторяться ровно n раз. Например, выражение **a{3}** будет соответствовать строкам, содержащим ровно три символа **a**.
5. {n,}: означает, что предыдущее символьное выражение должно повторяться n или более раз. Например, выражение **a{2,}** будет соответствовать строкам, содержащим два или более символов **a**.
6. {n,m}: означает, что предыдущее символьное выражение должно повторяться от n до m раз. Например, выражение **a{2,5}** будет соответствовать строкам, содержащим от двух до пяти символов **a**.

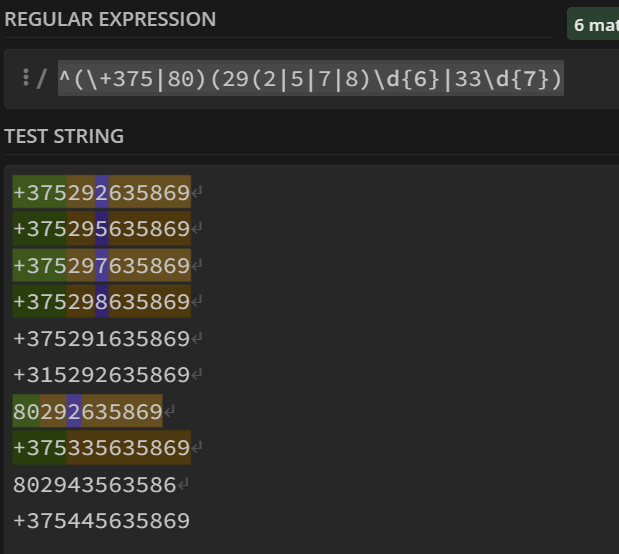
Это лишь некоторые из множества возможных кванторов в регулярных выражениях. Они позволяют создавать более гибкие и точные выражения для поиска и замены текста в строках.

7. Напишите регулярное выражение для проверки номера MTC (Velcom).



Регулярное выражение: “^(\+375|80)(29(2|5|7|8)\d{6}|33\d{7})$”

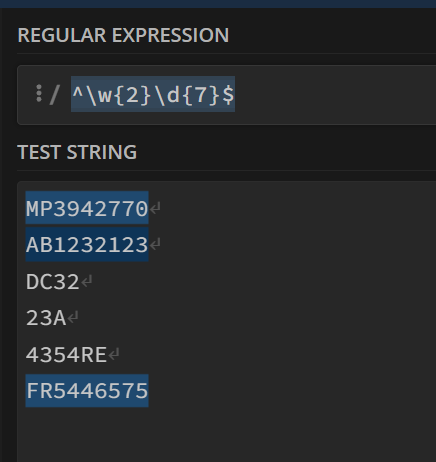
Проверка в сервисе [regex101: build, test, and debug regex](https://regex101.com/) :



1. Напишите регулярное выражение для проверки паспорта.

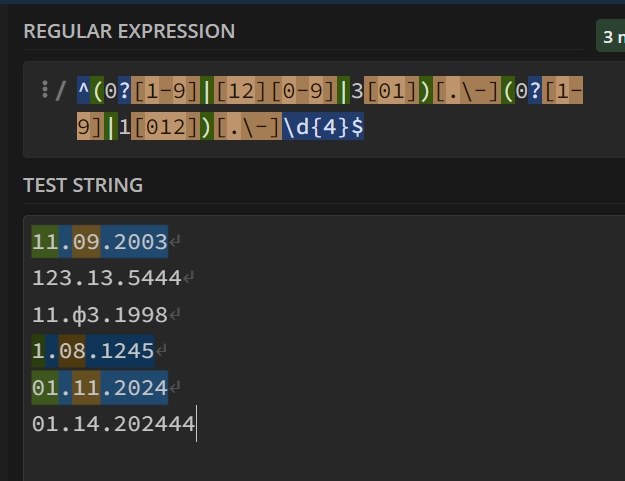


Регулярное выражение: ^\w{2}\d{7}$



1. Напишите регулярное выражение для проверки даты.

Регулярное выражение: ^(0?[1-9]|[12][0-9]|3[01])[.\-](0?[1-9]|1[012])[.\-]\d{4}$



\*[12] – это значит или 1 или 2

? – ноль или один раз

1. Напишите регулярное выражение для проверки УДК

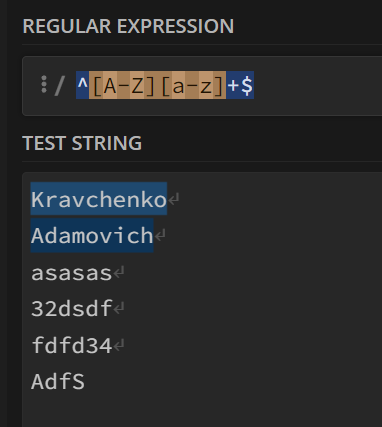
В интернете нет примеров формата УДК, GPT:

^[0-9]{2,3}(\.[0-9]+)?(\s\*[A-Z]{1,2})?(\:[0-9]+)?$

* **^[0-9]{2,3}** - начало строки с двух или трех цифр.
* **(\.[0-9]+)?** - опциональный десятичный разделитель, за которым следует одна или более цифр.
* **(\s\*[A-Z]{1,2})?** - опциональная последовательность из одного или двух буквенных символов в верхнем регистре, за которыми могут следовать один или несколько пробелов.
* **(\:[0-9]+)?** - опциональный двоеточный разделитель, за которым следует одна или более цифр.
* **$** - конец строки.

1. Напишите регулярное выражение для проверки Фамилии.

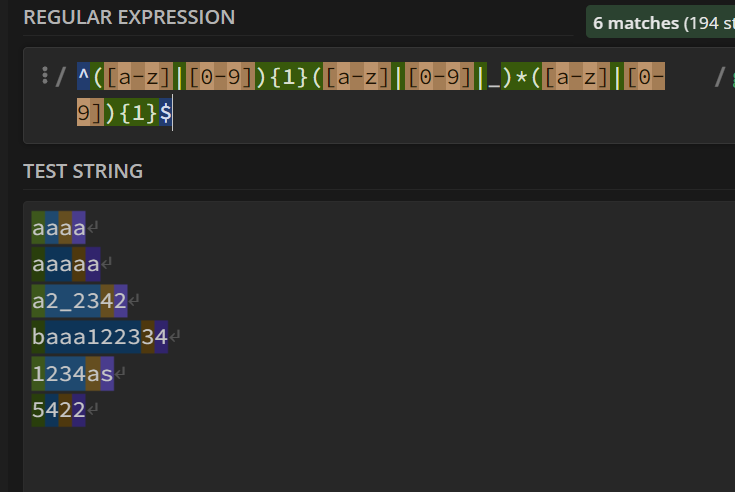
Регулярное выражение: ^[A-Z][a-z]+$



1. Напишите регулярное выражение для проверки пароля, логина.

Регулярное выражение: ^([a-z]|[0-9]){1}([a-z]|[0-9]|\_)\*([a-z]|[0-9]){1}$

Вводить можно буквы цифры и нижнее подчёркивание в пароль/логин



Разобрать:

**?(<img .\*?src\s\*=\s\*"")[^""]+(?="".\*?>)" :**

Данное регулярное выражение ищет значения атрибута "src" у тега <img> в HTML-документе.

Рассмотрим его более подробно:

**?<=<img .\*?src\s\*=\s\*"")** - это называется "positive lookbehind" и означает, что мы ищем символы, которые следуют непосредственно за выражением **<img** и атрибутом **src="**. Здесь используется **.\*?**, чтобы соответствовать любому количеству символов между **<img** и **src="**, не "жадным" способом (т.е. не захватывать больше, чем нужно).

**[^""]+** - это означает, что мы ищем любые символы, кроме двух кавычек (**"**), которые могут встречаться в значении атрибута **src**.

**(?="".\*?>)"** - это называется "positive lookahead" и означает, что мы ищем символы, которые предшествуют выражению **">** (закрывающий тег **>** для тега <img>). Здесь используется **.\*?**, чтобы соответствовать любому количеству символов между **"** и **>** для атрибута **src**, не "жадным" способом (т.е. не захватывать больше, чем нужно).

Таким образом, данное регулярное выражение находит значения атрибута **src** для тега **img** и игнорирует любые другие символы вокруг. Однако, стоит отметить, что регулярные выражения могут иметь свои ограничения в обработке HTML-документов, особенно если в документе есть сложные структуры или если атрибуты тегов записаны в необычных форматах.

**"[a-zA-Zа-яА-Я,-;:]{5,50}":**

Идёт вначале буква(заглавная\строчная\кириллица\латиница) – символов от 5 до 50

**/^[a-z0-9\_-]{3,16}$/:**

Начало строки – буквы строчные от a до z, цифры от 0 до 9, \_ и -. Символов от 3 до 16 и конец строки затем.

**/^#?([a-f0-9]{6}|[a-f0-9]{3})$/:**

Начало строки – решётка может не быть, а может быть одна. Идёт группа букв от a до f или цифры от 0 до 9 (ровно 6 их должно быть) ИЛИ Идёт группа букв от a до f или цифры от 0 до 9 (ровно 3 их должно быть) и конец строки затем.

**/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?$/:**

Начало строки – (https(может не быть, а может быть один раз), затем двоеточие и // ) в количестве 1 или 0. Затем (буквы d,a-z и символы .-) от 1 до бесконечности. Дальше точка, символы (a-z или точка) в количестве от 2 до 6. Ну и так далее, тут и так все понятно))

