

Регулярни изрази

Какво е това; кога и как се използват

Структури от данни
и алгоритми



Expression	<> PCRE ▾
<code>/([A-Z])\w+/g</code>	
Text	5 matches (0.2ms)
Learn, • Build, • & • Test • Regular • Expressions ▾	

Съдържание

- Какво са регулярните изрази
- Синтаксис: Литерали и метасимволи –
escaping последователности, метасимволи
за класове, количество, местоположение и др.
- Регулярни изрази в .NET
- Класът `Regex`
- Класовете `Match` и `MatchCollection`
- Търсене и извличане по регулярен израз
- Заместване в текст с регулярен израз
- Разделяне на низ по регулярен израз



Регулярни изрази

- Регулярните изрази са мощно средство за работа с текст:
 - **търсим** и **извличаме** информация от даден текст по даден шаблон
 - **валидираме** текстова информация
 - **заменяме** и **изтриваме** поднизове в даден текст чрез шаблони
- В .NET регулярните изрази имат синтаксиса на Perl 5
- Например:

разпознава произволен email

```
\b[\w.%+-]+@[ \w.-]+\.[a-zA-Z]{2,6}\b
```

Какво е регулярен израз?

- Регулярен израз описва някаква съвкупност от символни низове чрез специална граматика за описание на низовете
- Например:

`[0-1]+`

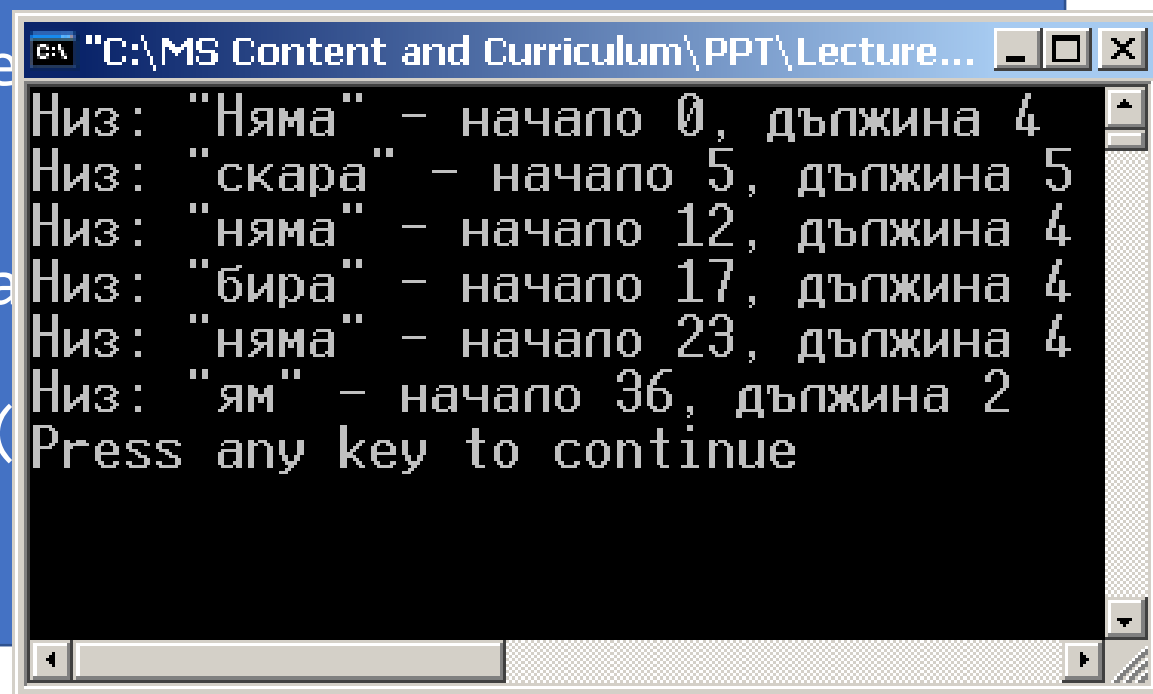
обозначава всички непразни низове, които се състоят само от цифрите 0 и 1, а низът

`088[0-9]{7}`

обозначава всички телефонни номера, които имат вида 088XXXXXX (където X е цифра)

Регулярни изрази в .NET – пример

```
static void Main(string[] args) {  
    string text = "Няма скара, няма бира, няма к'во да ям.";  
  
    // Регулярен израз за търсене на текст  
    string pattern = @"\w*ира|скара|\w*ям\w*";  
  
    Match match = Regex.Match(text, pattern);  
    while (match.Success) {  
        Console.WriteLine(  
            "Низ: \"{0}\" - начало {1}, дължина {2}",  
            match.Value, match.Index, match.Length);  
        match = match.NextMatch();  
    }  
}
```



```
C:\MS Content and Curriculum\PPT\Lecture...  
Низ: "Няма" - начало 0, дължина 4  
Низ: "скара" - начало 5, дължина 5  
Низ: "няма" - начало 12, дължина 4  
Низ: "бира" - начало 17, дължина 4  
Низ: "няма" - начало 23, дължина 4  
Низ: "ям" - начало 36, дължина 2  
Press any key to continue
```

Валидация с регулярни изрази

- Регулярните изрази са много удобни за **валидация** на входни данни:

```
string email = "test-1.p46@ala-bala.somehost.somewhere.bg";  
string regex = @"^([a-zA-Z0-9_\-][a-zA-Z0-9_\-\.]{0,49})" +  
    @"@((([a-zA-Z0-9][a-zA-Z0-9\-\-]{0,49}\.))+[a-zA-Z]{2,4})$";  
bool valid = Regex.IsMatch(email, regex);  
Console.WriteLine(valid);
```

- Валидни email адреси:
[nyakoi-lom@abv.bg](#), [-123--@usa.net](#), [test.test123@en.some-host.12345.com](#)
- Невалидни email адреси:
[.ala.@bala.com](#), [user@.test.ru](#), [user@test.ru.](#), [alabala@](#), [user@host](#), [@eu.net](#)

Езикът на регулярните изрази

- С регулярните изрази **описваме низовете, които търсим** в даден текст
- Синтаксисът на регулярните изрази се състои от:
 - **литерали** - части от низове, които търсим
 - **метасимволи** - команди, които задават специални правила за търсене
- Например, в регулярния израз

```
\w*ира|скара|\w*ям\w*
```

литерали са "ира", "скара" и "ям", а метасимволи са "\w", "*" и "|"

Категории метасимволи

- Escaping последователности, например `*`
- Класове от символи, например `[a-zA-Z]`
- За количеството (quantifiers), например `*` и `+`
- За местоположението в текста, например `\b`
- За алтернативен избор, например `|` (логическо "или")
- Групиращи метасимволи, например `([0-9]+)`
- Други - например `#` (за коментари), `$1` (заместващи) и т.н.

Escaping последователности

- `\t` – табулация
- `\r` – символ за връщане на каретката CR (0x0D)
- `\n` – символ за нов ред LF (0x0A)
- `\xXX` – символ с ASCII код XX (шестнайсетично)
- `\uXXXX` – Unicode символ с номер XXXX (шестнайсетично)
- `\\` – символ `\`
- `*` – символ `*`
- `\+` – символ `+`

Класове от символи

- `.` – обозначава произволен символ без `\n`
- `(.|\s)` – обозначава произволен символ
- `[СИМВОЛИ]` – обозначава произволен символ от изброените
 - Пример: `[01]` обозначава цифрата 0 или 1
- `[^СИМВОЛИ]` – обозначава всеки символ, който не е сред изброените
 - Пример: `[^<>\\]` обозначава всеки символ без `<`, `>` и `\`
- `[charX-charY]` – обозначава символ в зададения интервал
 - Пример: `[0-9A-F]` обозначава всеки символ, който е цифра или латинска буква между A и F

Класове от символи

- `\w` – буквите, цифрите и символа `_` (за всички езици от Unicode)
- `\W` – всички символи с изключение на буквите, цифрите и `_`
- `\s` – символите за празно пространство (интервал, табулация, нов ред, ...)
- `\S` – символите, които не са празно пространство
- `\d` – десетичните цифри `[0–9]`
- `\D` – всички символи, които не са десетични цифри

Метасимволи за количество

- $*$ – нула или повече срещания
 - Пример: $[01]^*$ обозначава всички символни низове, съставени от цифрите 0 или 1, включително празния низ
- $+$ – едно или повече срещания
 - Пример: $[A-Z]^+$ задава непразните символни низове, съставени от главни латински букви
- $?$ – нула или едно срещания
 - Пример: $[A-Z]?$ обозначава главна латинска буква или празен низ
- $\{n\}$ – точно n срещания
 - Пример: $[0-9]\{3\}$ обозначава последователност от точно 3 цифри

Метасимволи за количество

- $\{n, \}$ – поне n срещания
 - Пример: $[0-9]\{5, \}$ обозначава последователност от поне 5 цифри
- $\{n, m\}$ – поне n и най-много m срещания
 - Пример: $[0-9]\{2, 4\}$ обозначава последователност от 2, 3 или 4 цифри
- $*?$ – нула или повече срещания, но най-малкият възможен брой
- $+?$ – едно или повече срещания, но най-малкият възможен брой
- $\{n, \}?$ – поне n срещания, но най-малкият възможен брой

Метасимволи за местоположение

- `\b` – търсене само в началото или края на дума – на границата между символите `\w` и `\W`, но в рамките на `\w`
 - Пример: `\b\w*бир\w*\b` обозначава всички думи, съдържащи като подниз "бир", например "бира", "обирам" но не и "налей ми биричка"
 - Пример: `.*?ра\b` обозначава всички най-къси поднизове, завършващи на "ра", например "кака Мара" и "дай бира", но не и "бира няма" и "подай ми бирата"
- `\B` – търсене само в средата на думата (без началото и края)
 - Пример: `\w*бира\B` ще намери "бира" в текста "Къде ми е бирата?", но няма да намери нищо в текста "Налей ми бира, моме ле!"

Метасимволи за местоположение

- `\A` – указание за търсене само в началото на подадения текст (задава се преди низа)
 - Пример: `\Абира` ще намери "бира" в текста "бирата е хладна", но няма да намери нищо в текста "Живот без бира не е живот!"
- `\Z` – указание за търсене само в края на текста (задава се след низа)
 - Пример: `бира\Z` ще намери "бира" в текста "налей бира", но няма да намери нищо в текста "бирата е хладна"
- `^` – търсене само в началото на текста (в режим multi-line и в началото на всеки ред)
- `$` – търсене само в края на текста (в режим multi-line и в края на всеки ред)

Други метасимволи

- **Метасимволи за избор:**
 - $A | B$ – задава алтернативен избор между регулярните изрази A и B
 - Пример: бира | скара ще намери "бира" в текста "Студена ли е бирата?", а в текста "Къде е скарата?" ще намери "скара"
- **Метасимволи за задаване на групи:**
 - (група) , $(?<\text{име}>\text{група})$ – задава група в регулярния израз (без име или с име)
 - Пример: $\backslash s^* (?<\text{name}>\backslash w+) \backslash s^* = \backslash s^* (\backslash d+)$
 - Групите се използват за логическо отделяне на части от регулярния израз и могат да имат имена

Регулярните изрази в .NET Framework

- Намират се в `System.Text.RegularExpressions`:
 - Класът `Regex`
 - очаква регулярен израз
 - има методи за търсене, заместване и разделяне на низове чрез този израз
 - има и статични методи за всички основни операции
 - Класът `Match`
 - съдържа описание на едно съвпадение (стойност, начална позиция и дължина), получено в резултат от търсене с регулярен израз
 - позволява намиране на следващи съвпадения от търсенето, ако има такива

Регулярните изрази в .NET Framework

- Класът `MatchCollection`
 - съдържа списък от съвпадения (получени в резултат от търсене)
- Класът `Group`
 - представлява група от символи, съдържаща се в дадено съвпадение (`Match`). В едно съвпадение може да има няколко групи
- Класът `GroupCollection`
 - съдържа списък от групи, съдържащи се в дадено съвпадение
- Има и други помощни класове

Класът Regex

- Най-важният клас за работа с регулярни изрази в .NET
- Може да се ползва по два начина:
 1. Създава се обект от класа, като в конструктора се подава регулярен израз, после се извикват методите му за обработка на текст (`IsMatch`, `Match`, `Matches`, `Replace`, `Split`)
 2. Използват се статичните методи на класа (`IsMatch`, `Match`, `Matches`, `Replace`, `Split`), на които се подава текста за обработка и регулярен израз, който ще се използва
- Първият подход е по-ефективен, ако с един и същ израз ще се обработват последователно няколко текста

Класът Regex – методи и свойства

- **IsMatch(text, pattern)** – проверява дали в даден текст се среща поне един подниз, който съответства на даден регулярен израз
- **Match(text, pattern)** – търси зададения регулярен израз в зададения текст и връща първото съвпадение като **Match** обект
- **Matches(text, pattern)** – търси зададения регулярен израз в зададения текст и връща **MatchCollection** от всички съвпадения
- **Replace(text, pattern, replacement)** – замества всички срещания за даден регулярен израз в даден текст със заместващ текст, който може да съдържа части от намерените съвпадения (групи в рег. израз)

Класът Regex – методи и свойства

- **string[] Split(text, pattern)** – разделя даден низ на части по даден регулярен израз
- **Escape(text)** – замества всички специални символи за езика на регулярните изрази с еквивалентни escaping последователности
- **Unescape(text)** – обратното на **Escape()**
- **GetGroupNames()** – връща масив от имената на групите, дефинирани в даден регулярен израз (групите без имена автоматично получават служебно име)
- **Options** – свойство от тип **RegexOptions**, което задава някои настройки на израза (като **Singleline**, **Multiline**, **IgnoreCase** и **IgnorePatternWhitespace**)

Regex.IsMatch – пример

```
static bool IsPositiveInteger(string aNumber) {  
    Regex numberRegex = new Regex(@"\A[1-9][0-9]*\Z");  
    return numberRegex.IsMatch(aNumber);  
}  
  
static void Check(string aText) {  
    Console.WriteLine("{0} - {1}", aText,  
        IsPositiveInteger(aText) ? "positive integer" :  
        "NOT a positive integer");  
}  
  
static void Main(string[] args) {  
    Check("123456"); // 123456 - positive integer  
    Check("15 16"); // 15 16 - NOT a positive integer  
}
```

Целите
положителни
числа започват с
цифра от 1 до 9
и после имат 0
или повече цифри
в края

Regex.Match – пример

```
static void Main(string[] args) {
    string text = @"<html>This is a hyperlink:
    <a href=""javascript:'window.close()'">
    close the window</a><br> ... and one more link: <a
    target=""_blank"" href=/main.aspx class='link'> <b>
    main page</b> </a>< a href = 'http://www.nakov.com'
    > <img src='logo.gif'>Nakov's home site < /a >";

    string hrefPattern = @"<\s*a\s[^\>]*\bhref\s*=\s*" +
        @"('^[^']*'|""^[^"""]*""|\S*)[^\>]*>" +
        @"(.\s)*?<\s*/a\s*>";

    Match match = Regex.Match(text, hrefPattern);
    while (match.Success) {
        Console.WriteLine("{0}\n\n", match);
        match = match.NextMatch();
    }
}
```

Regex.Matches – пример

```
static void Main() {  
    // Регулярен израз за търсене на думи на кирилица  
    Regex regex = new Regex(@"\b[A-Яa-я]+\b");  
  
    String text = "The Bulgarian word 'бира' (beer) often" +  
                  " comes with the word 'скара' (grill).";  
  
    MatchCollection matches = regex.Matches(text);  
    foreach (Match match in matches) {  
        Console.WriteLine("{0}:{1} ", match); // бира скара  
    }  
}
```


Класът `Match` – методи и свойства

- **Success** – връща дали търсенето е намерило нещо
- **Value** – връща стойността на съвпадението
- **Index** – връща позицията на съвпадението в текста
- **Length** – връща дължината на съвпадението
- **NextMatch()** – предизвиква продължаване на търсенето от края на текущата съвпадение и връща следващото съвпадение (ако има)
- **Groups** – връща групите, съдържащи се в съвпадението във вид на `GroupCollection`
- **Captures** – връща `CaptureCollection` от `Capture` обектите, образуващи съвпадението

Regex.Match – пример

```
static void Main() {
    // Регулярен израз за търсене на думи на латиница
    Regex regex = new Regex(@"\b[A-Za-z]+\b");

    String text = "Бирените историците смятат, че същинският " +
                  "хмел (Humulus lupulus) влязъл трайно в " +
                  "пивоварството едва през IX век.";

    Match match = regex.Match(text);
    while (match.Success) {
        Console.WriteLine("{0}:{1} ", match, match.Index);
        match = match.NextMatch();
    }    // Резултат: Humulus:47 lupulus:55 IX:103
}
```

Работа с групи

- **Групите** в регулярните изрази дават възможност **за извличане** и обработка **на отделните части** от изразите
 - Групите се задават със скоби и могат да имат имена
 - Групите могат да се влагат една в друга
- За извличане на всички групи от дадено съвпадение (`Match`) се използва свойството **`Groups`**, което връща `GroupCollection`
 - Групите, за които не са зададени имена, могат да се извличат по номер
 - Номерацията става по реда на отварящите скоби

Работа с групи – пример

```
String text = "gosho 62.44.18.124 02:44:50\n" +
               "root 193.168.22.18 22:12:38";
string pattern = <потребител> <IP адрес> <време в системата>
                 @"(?<name>\S+)\s+(?<ip>[0-9\.]+\s+(?<time>[0-9:]+)";
MatchCollection matches = Regex.Matches(text, pattern);
foreach (Match match in matches) {
    Console.WriteLine("name={0,-8} ip={1,-16} time={2}",
                      match.Groups["name"], match.Groups["ip"],
                      match.Groups["time"]);
}
```

Regex.Replace – пример

```
static void Main() {
    String text = "Here is the link:<br>" +
        "[URL=http://www.devgb.org]БАРС[/URL]<br>\n" +
        "and the logo:[URL=http://www.devgb.org][IMG]\n" +
        "http://www.devgb.org/basd-logo.png[/IMG][ /URL]\n";
    string pattern = @"\[URL=(?<url>[^\]]+)\]" +
        @"(?<content>(.\s)*?)\[ /URL\]";
    string newPatt = "<a href=\"${url}\">${content}</a>";
    string newText =
        Regex.Replace(text, pattern, newPatt);
    Console.WriteLine(newText);
}
```

За да заместим тагове
[URL=...] ... [/URL] с
HTML хипервръзки
 ...

Regex.Split – пример

- Извличане на email адреси от списък с най-различни разделители:

```
String text = "zuzi@mail.bg;; cici@abv.bg, " +  
    "bob@mail.bg\n\nfn12345@fmi.uni-sofia.bg\n" +  
    "    mente@eu.int | , , ;;; gero@dir.bg";  
string splitPattern = @"[;|,|\s|\|]+";  
string[] emails = Regex.Split(text, splitPattern);  
Console.WriteLine(String.Join(", ", emails));
```

```
// Резултат: zuzi@mail.bg, cici@abv.bg, bob@mail.bg,  
// fn12345@fmi.uni-sofia.bg, mente@eu.int, gero@dir.bg
```

- Ако използваме групи в разделящия регулярен израз, те участват в резултата:

```
string[] parts = Regex.Split("скара-бира", "(-)");  
// parts = {"скара", "-", "бира"}
```

Настройки с `Regex.Options`

- **IgnoreCase** – задава търсене, което игнорира регистъра на буквите (малки/главни)
- **Multiline** – задава режим за търсене "multi-line". Метасимволите `^` и `$` в този режим означават начало и край на ред
- **Singleline** – задава режим за търсене "single-line". Метасимволът `.` в този режим има значение "всеки символ, включително `\n`"
- **IgnorePatternWhitespace** – игнорира празното пространство и коментарите (`# ...`) в рег. израз

RegexOptions – примери

```
String text = "Бирата намалѐ. Дайте още бира!";
string pattern = @"\bбир\w*\b";
MatchCollection matches = Regex.Matches(text, pattern, RegexOptions.IgnoreCase);
foreach (Match match in matches)
{
    Console.WriteLine("{0} ", match);
}
// Заради опцията IgnoreCase резултатът е: Бирата бира
```

```
String text = "Бирата намалѐ.\nДайте още бира!";
string pattern = @"^\w+"; \ \ търсим думи в началото на ред
MatchCollection matches = Regex.Matches(text, pattern, RegexOptions.Multiline);
foreach (Match match in matches)
{
    Console.WriteLine("{0} ", match);
}
// Заради опцията Multiline резултатът е: Бирата Дайте
```


Escaping на регулярен израз

- При динамично конструиране на регулярен израз трябва да се ескаре-ват всички данни, идващи от външни източници
- Ето например как можем да търсим къде се среща дадена дума в даден текст:

```
String text = @"Всички форми на думата 'бира' в даден" +  
    @" текст можем да намерим с регулярния израз " +  
    @"\b(Б|б)ир(((а|ичка)(та)?)|((и|ички)(те)?))\b.";  
string word = Console.ReadLine();  
String pattern = @"\b" + Regex.Escape(word) + @"\b";  
Match match = Regex.Match(text, pattern);
```

- Какво ще намери този код ако бяхме пропуснали escaping-а и потребителят въведе за търсене "\w*" или ". *"?
- Каква друга грешка има в кода? Какво ще стане ако потребителят въведе празен низ?

Кога да ползваме рег. изрази?

- Регулярните изрази **се поддържат трудно!**
 - Ако използвате сложни регулярни изрази, помислете дали ще можете с лекота да ги промените след 3 месеца! Ами след 1 година?
 - А вашите колеги ще могат ли при нужда да ги променят?
- Като правило **избягвайте сложни регулярни изрази**
 - Разбивайте проблема на части и пишете по-прости регулярни изрази за всяка част
 - Например за изваждане на всички изречения с главни букви първо извадете изреченията, а след това проверете дали са от главни букви

Ами ефективността?

- Ефективността на регулярните изрази може да бъде **трагично ниска**, ако бъдат използвани неправилно
 - Например следният код може да умори за доста време дори завидно бърза машина:

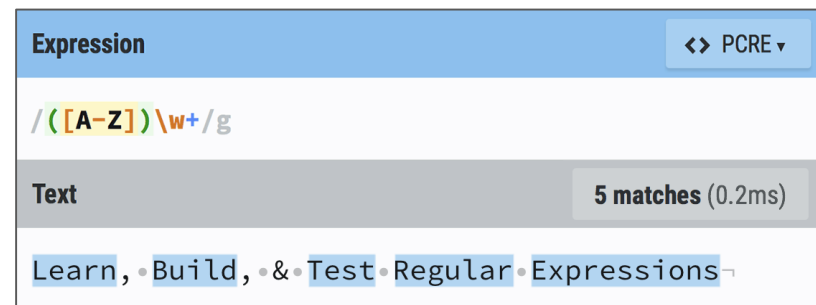
```
String text = "aaabacabababaccbacbcbccacbcbbaccccc";  
string pattern = @"(\w*ab|\w*ac|\w*aa|\w*c)*cccccc";  
Match m = Regex.Match(text, pattern);
```

Търсите готови изрази?

- Не е нужно да знаете всичко за регулярните изрази
- Има сайтове, където можете да намерите на готово каквото ви трябва:
 - Regular Expressions Library – <http://www.regexlib.com/>
 - 3 Leaf: .NET Regular Expression Repository – <http://www.3leaf.com/resources/articles/regex.aspx>
- Има и инструменти за създаване и тестване на рег. изрази:
 - The Regulator – <http://royo.is-a-geek.com/serializable/regulator/>
 - The Regex Coach – <http://www.weitz.de/regex-coach/>
 - RegEx Storm - <http://regexstorm.net/reference>
 - RegExpr - <https://regexr.com/>
 - Regular expressions 101 - <https://regex101.com/>

Какво научихме днес?

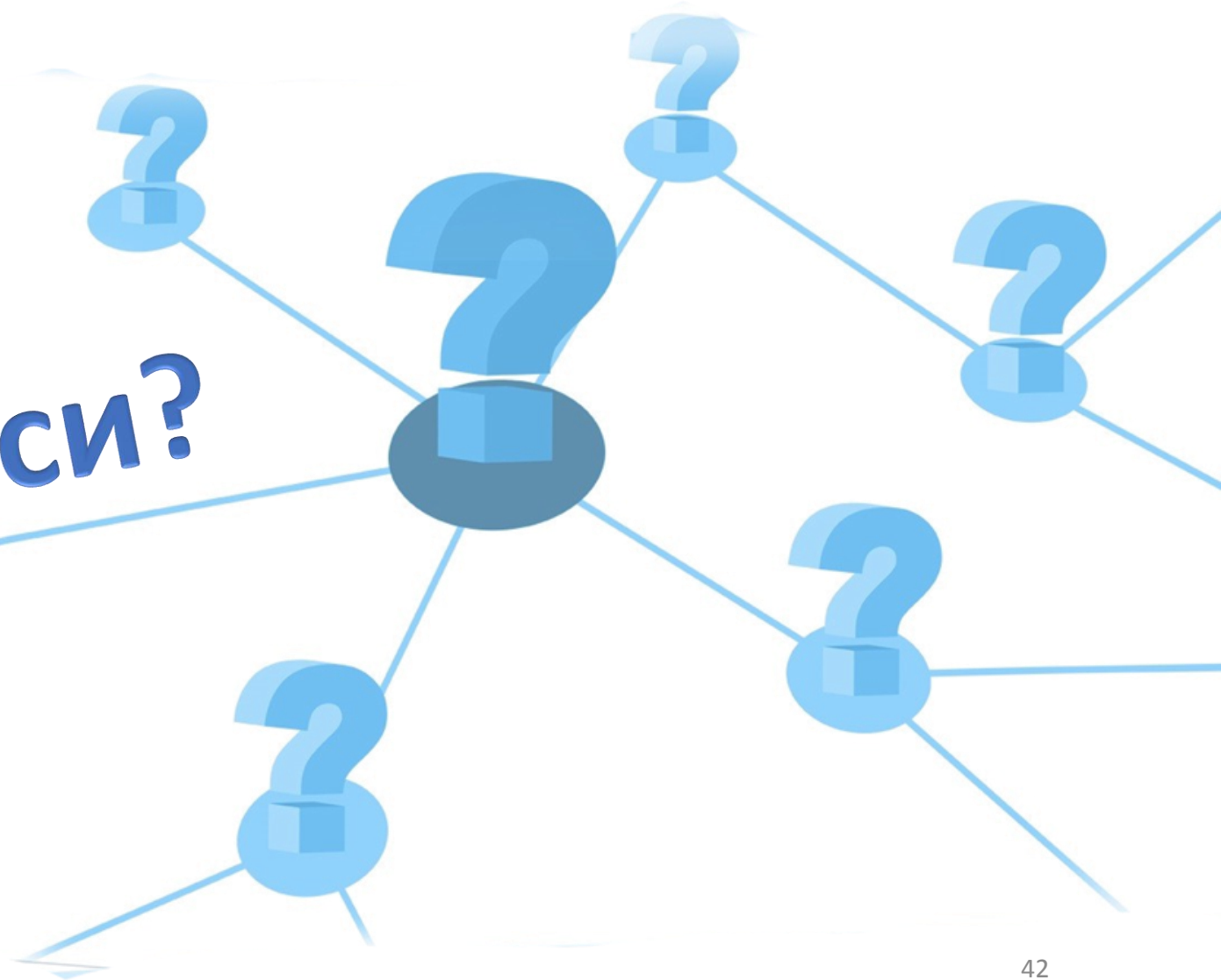
- **Регулярните изрази** са мощно средство за работа с текст (търсене и заместване, извличане, валидиране на текст и други)
- С тях **описваме низовете, които търсим** в даден текст, с помощта на специален език
- Синтаксисът му се състои от:
 - **литерали** - частите от низове, които търсим
 - **метасимволи** - команди, които задават специални указания и правила за търсене – ескаре последователности, класове символи, количество, местоположение и др.



Регулярни изрази



Въпроси?



Договор за ползване

Този курс (слайдове, примери, задачи и др.) се разпространяват под свободен лиценз "[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International](https://creativecommons.org/licenses/by-nc-sa/4.0/)"



Базиран е на [презентация за регулярните изрази](#) на Светлин Наков и на [друга](#), направена съвместно с Георги Пенков.