

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

# Wyrażenia regularne. Leksykon kieszonkowy

Autor: Tony Stubblebine

Tłumaczenie: Piotr Rajca

ISBN: 83-7361-075-8

Tytuł oryginału: [Regular Expressions Pocket Reference](#)

Format: B5, stron: 128



Wyrażenia regularne są niezwykle potężnym mechanizmem służącym do dopasowywania i manipulowania tekstami. Choć są one dostępne w wielu nowoczesnych językach programowania, to jednak w każdym z nich posiadają one nieco inne możliwości, a subtelne różnice w ich składni sprawiają, że nie są one w pełni zgodne z wyrażeniami stosowanymi w innych językach. Wiele języków programowania implementuje wyrażenia regularne, jednak żaden z nich nie robi tego w identyczny sposób.

Książka „Wyrażenia regularne. Leksykon kieszonkowy” zawiera krótką prezentację pojęć związanych z wyrażeniami regularnymi oraz omówienia ich najczęściej spotykanych implementacji dysponujących największymi możliwościami. Nie będziesz już musiał rozszyfrowywać tajemniczych lub chaotycznych materiałów informacyjnych! Niniejsza książka zawiera tabele i porady dotyczące stosowania wyrażeń regularnych w językach Perl, Java, Python, C# (i środowisku .NET), PHP, JavaScript, w bibliotece PCRE oraz kilku programach (vi, awk, egrep oraz sed).

Ten leksykon można potraktować jako dopełnienie doskonałej i wyczerpującej książki poświęconej wyrażeniom regularnym – „Wyrażenia regularne”, autorstwa Jeffreya E. F. Friedla. Dzięki niemu będziesz mógł szybko określić składnię oraz szczególne cechy wyrażeń regularnych w dowolnym używanym języku programowania.



# Spis treści

<b>O książce .....</b>	<b>5</b>
<b>Przedstawienie wyrażeń regularnych oraz zagadnień dopasowywania wzorców .....</b>	<b>7</b>
Metaznaki, tryby oraz konstrukcje .....	10
Reprezentacja znaków .....	10
<b>Perl 5.8 .....</b>	<b>22</b>
Obsługiwane metaznaki .....	22
Operatory wyrażeń regularnych .....	27
Obsługa Unicode .....	30
Przykłady .....	31
Inne źródła informacji .....	32
<b>Java (java.util.regex) .....</b>	<b>33</b>
Obsługiwane metaznaki .....	33
Klasy i interfejsy związane z wykorzystaniem wyrażeń regularnych .....	38
Obsługa Unicode .....	44
Przykłady .....	45
Inne źródła informacji .....	47
<b>.NET i C# .....</b>	<b>48</b>
Obsługiwane metaznaki .....	48
Klasy i interfejsy związane z wykorzystaniem wyrażeń regularnych .....	53
Obsługa Unicode .....	59
Przykłady .....	59
Inne źródła informacji .....	62
<b>Python .....</b>	<b>62</b>
Obsługiwane metaznaki .....	62
Obiekty i funkcje modułu re .....	66

Obsługa Unicode .....	71
Przykłady .....	72
Inne źródła informacji .....	73
<b><i>Biblioteka PCRE</i></b> .....	<b>73</b>
Obsługiwane metaznaki .....	79
PCRE API .....	79
Obsługa Unicode .....	83
Przykłady .....	84
Inne źródła informacji .....	88
<b><i>PHP</i></b> .....	<b>88</b>
Obsługiwane metaznaki .....	88
Funkcje obsługi wyrażeń regularnych .....	93
Przykłady .....	96
Inne źródła informacji .....	98
<b><i>Edytor vi</i></b> .....	<b>98</b>
Obsługiwane metaznaki .....	98
Dopasowywanie wzorców .....	102
Przykłady .....	103
Inne źródła informacji .....	104
<b><i>JavaScript</i></b> .....	<b>104</b>
Obsługiwane metaznaki .....	104
Metody i obiekty związane z wykorzystaniem wyrażeń regularnych .....	107
Przykłady .....	111
Inne źródła informacji .....	113
<b><i>Programy obsługiwane z wiersza poleceń</i></b> .....	<b>113</b>
Obsługiwane metaznaki .....	113
Inne źródła informacji .....	119
<b><i>Skorowidz</i></b> .....	<b>121</b>

## Java (*java.util.regex*)

W Java 1.4 obsługa wyrażeń regularnych została zaimplementowana w formie pakietu `java.util.regex`, opracowanego przez firmę Sun. Choć istnieją także inne pakiety obsługujące wyrażenia regularne, przeznaczone dla wcześniejszych wersji języka, to jednak standardem stanie się zapewne pakiet firmy Sun. W pakiecie tym wykorzystywany jest tradycyjny mechanizm dopasowywania NFA. Wyjaśnienie zasad działania tego mechanizmu można znaleźć w rozdziale pod tytułem „Przedstawienie wyrażeń regularnych oraz zagadnień dopasowywania wzorców”.

### Obsługiwane metaznaki

Pakiet `java.util.regex` obsługuje metaznaki oraz metasekwencje przedstawione w tabelach od 10. do 14. Bardziej szczegółowe opisy poszczególnych metaznaków można znaleźć w części zatytułowanej „Metaznaki, tryby oraz konstrukcje”.

Tabela 10. Reprezentacje znaków

Sekwencja	Znaczenie
<code>\a</code>	Alarm (dzwonek)
<code>\b</code>	Znak cofnięcia; <code>x08</code> , obsługiwany wyłącznie w klasie znaków
<code>\e</code>	Znak ESC, <code>x1B</code>
<code>\n</code>	Znak nowego wiersza, <code>x0A</code>

<code>\r</code>	Znak powrotu karetki, <code>x0D</code>
-----------------	--

Tabela 10. Reprezentacje znaków — ciąg dalszy

Sekwencja	Znaczenie
<code>\f</code>	Znak wysunięcia kartki, <code>x0C</code>
<code>\t</code>	Znak tabulacji poziomej, <code>x09</code>
<code>\0ósemkowa</code>	Znak reprezentowany przez jedno-, dwu- lub trzycyfrową liczbę ósemkową
<code>\xszesnastkowa</code>	Znak reprezentowany przez jedno- lub dwucyfrową liczbę szesnastkową
<code>\uszesnastkowa</code>	Znak reprezentowany przez czterocyfrowy kod szesnastkowy Unicode
<code>\cznak</code>	Znak sterujący o podanej nazwie

Tabela 11. Klasy znaków oraz skrótowe zapisy klas

Klasa	Znaczenie
<code>[...]</code>	Pojedynczy znak podany lub zawierający się w określonym zakresie
<code>[^...]</code>	Pojedynczy znak, który nie został podany lub nie zawiera się w określonym zakresie
<code>.</code>	Dowolny znak za wyjątkiem znaku nowego wiersza (chyba że stosowany jest tryb DOTALL)
<code>\w</code>	Znak mogący tworzyć wyrazy; <code>[a-zA-Z0-9_]</code>
<code>\W</code>	Znak, który nie może tworzyć wyrazów; <code>[^a-zA-Z0-9_]</code>
<code>\d</code>	Cyfra; <code>[0-9]</code>
<code>\D</code>	Dowolny znak nie będący cyfrą; <code>[^0-9]</code>
<code>\s</code>	odstęp; <code>[\t\n\f\r\x0B]</code>
<code>\S</code>	Dowolny znak nie będący odstępem; <code>[^\t\n\f\r\x0B]</code>
<code>\p{właściwość}</code>	Znak należący do określonej klasy znaków POSIX bądź właściwości lub bloku Unicode
<code>\P{właściwość}</code>	Znak, który nie należy do określonej klasy znaków POSIX ani właściwości lub bloku Unicode

Tabela 12. Punkty zakotwiczenia i warunki zerowej długości

Sekwencja	Znaczenie
^	Początek łańcucha znaków lub, w trybie wielowierszowym (MULTILINE), miejsce położone bezpośrednio za znakiem nowego wiersza
\A	Początek łańcucha znaków, niezależnie od używanego trybu dopasowywania
\$	Koniec łańcucha znaków lub, w trybie wielowierszowym (MULTILINE), miejsce położone bezpośrednio przed dowolnym znakiem nowego wiersza
\Z	Koniec łańcucha znaków lub miejsce położone przed znakiem nowego wiersza kończącym wejściowy łańcuch znaków, niezależnie od używanego trybu dopasowywania
\z	Koniec łańcucha znaków, niezależnie od używanego trybu dopasowywania
\b	Granica słowa
\B	Dowolne miejsce, które nie jest granicą słowa
\G	Początek aktualnego wyszukiwania
(?=...)	Pozytywne przewidywanie
(?!...)	Negatywne przewidywanie
(<=...)	Pozytywne przewidywanie wsteczne
(<!...)	Negatywne przewidywanie wsteczne

Tabela 13. Komentarze i modyfikatory trybu

Modyfikator (sekwencja)	Znak trybu	Znaczenie
Pattern.UNIX_LINES	d	Traktuje znak \n jako jedyny znak końca wiersza
Pattern.DOTALL	s	Kropka (.) odpowiada dowolnemu znakowi, w tym także znakowi zakończenia wiersza
Pattern.MULTILINE	m	Metaznaki ^ oraz \$ pasują do miejsc położonych tuż obok osadzonych znaków końca wiersza
Pattern.COMMENTS	x	Ignoruje odstępę i pozwala na umieszczanie komentarzy w wyrażeniu regularnym

Pattern.CASE_INSENSITIVE	i	Podczas dopasowywania nie będzie uwzględniana wielkość liter należących do kodu ASCII
--------------------------	---	---

Tabela 13. Komentarze i modyfikatory trybu — ciąg dalszy

Modyfikator (sekwencja)	Znak trybu	Znaczenie
Pattern.UNICODE_CASE	u	Podczas dopasowywania nie będzie uwzględniana wielkość liter Unicode
Pattern.CANON_EQ		Tryb „kanonicznej równoważności” Unicode, w którym znak oraz sekwencja składająca się ze znaku bazowego i znaków łączących o takiej samej reprezentacji wizualnej są traktowane jako identyczne
(?tryb)		Włącza podane tryby (idmsux) w dalszej części podwyrażenia
(?-tryb)		Wyłącza podane tryby (idmsux) w dalszej części podwyrażenia
(?tryb:...)		Włącza podane tryby (idmsux) w wyrażeniu podanym pomiędzy dwukropkiem i nawiasem zamykającym
(?-tryb:...)		Wyłącza podane tryby (idmsux) w wyrażeniu podanym pomiędzy dwukropkiem i nawiasem zamykającym
#...		W trybie /x sprawia, że wszystkie znaki do końca wiersza będą traktowane jako komentarz

Tabela 14. Grupowanie, przechwytywanie, konstrukcje warunkowe i sterowanie

Sekwencja	Znaczenie
(...)	Grupuje podwzorce, przechwytyje pasujące do nich łańcuchy znaków i zapamiętuje je w metaznakach \1, \2, ... oraz \$1, \$2, ...
\n	Zawiera tekst odpowiadający <i>n</i> -tej grupie przechwytywanej
\$n	W łańcuchu zamiennika zawiera tekst dopasowany do <i>n</i> -tej grupy przechwytywanej

(?:...)	Grupuje podwzorce, lecz nie powoduje przechwycenia i zapamiętania pasujących do nich łańcuchów znaków
(?>...)	Nie pozwala na powtórne wykorzystanie tekstu pasującego do podwzorca

*Tabela 14. Grupowanie, przechwytywanie, konstrukcje warunkowe i sterowanie — ciąg dalszy*

<b>Sekwencja</b>	<b>Znaczenie</b>
... ...	Sprawdza alternatywne podwzorce
*	Dopasowuje podwzorec 0 lub więcej razy
+	Dopasowuje podwzorec 1 lub więcej razy
?	Dopasowuje podwzorec 1 lub 0 razy
{ <i>n</i> }	Dopasowuje podwzorec dokładnie <i>n</i> razy
{ <i>n</i> ,}	Dopasowuje podwzorec co najmniej <i>n</i> razy
{ <i>x</i> , <i>y</i> }	Dopasowuje podwzorec co najmniej <i>x</i> razy, jednak nie więcej niż <i>y</i> razy
*?	Dopasowuje 0 lub więcej powtórzeń podwzorca, przy czym wybierana jest najmniejsza możliwa liczba powtórzeń
+?	Dopasowuje 1 lub więcej powtórzeń podwzorca, przy czym wybierana jest najmniejsza możliwa liczba powtórzeń
??	Dopasowuje 0 lub 1 powtórzenie podwzorca, przy czym wybierana jest najmniejsza możliwa liczba powtórzeń
{ <i>n</i> ,}?	Dopasowuje podwzorec co najmniej <i>n</i> razy, przy czym wybierana jest najmniejsza możliwa liczba powtórzeń
{ <i>x</i> , <i>y</i> }?	Dopasowuje podwzorec co najmniej <i>x</i> razy, jednak nie więcej niż <i>y</i> razy, przy czym wybierana jest najmniejsza możliwa liczba powtórzeń
*+	Dopasowuje 0 lub więcej powtórzeń podwzorca, przy czym nigdy nie jest realizowane nawracanie
++	Dopasowuje 1 lub więcej powtórzeń podwzorca, przy czym nigdy nie jest realizowane nawracanie
?+	Dopasowuje 0 lub 1 powtórzenie podwzorca, przy czym nigdy nie jest realizowane nawracanie
{ <i>n</i> }+	Dopasowuje podwzorec co najmniej <i>n</i> razy, przy czym nigdy nie jest realizowane nawracanie
{ <i>n</i> ,}+	Dopasowuje podwzorec co najmniej <i>n</i> razy, przy czym nigdy nie jest realizowane nawracanie

## *Klasy i interfejsy związane z wykorzystaniem wyrażeń regularnych*

W języku Java 1.4 wprowadzono dwie podstawowe klasy związane z obsługą wyrażeń regularnych — `java.util.regex.Pattern` oraz `java.util.regex.Matcher`, jeden wyjątek — `java.util.regex.PatternSyntaxException` oraz nowy interfejs — `CharSequence`. Ponadto firma Sun zaktualizowała klasę `String` — aktualnie implementuje ona interfejs `CharSequence` i udostępnia podstawowe metody związane z wykorzystaniem wyrażeń regularnych i dopasowywaniem wzorców. Obiekty `Pattern` to skompilowane wyrażenia regularne, które można dopasowywać do wielu różnych łańcuchów znaków. Z kolei obiekt `Matcher` to wynik dopasowania jednego obiektu `Pattern` do konkretnego łańcucha znaków (lub dowolnego obiektu implementującego interfejs `CharSequence`).

Znaki odwrotnego ukośnika umieszczone w literałach znakowych definiujących wyrażenie regularne należy odpowiednio oznaczać (dodatkowym znakiem odwrotnego ukośnika). A zatem `\n` (znak nowego wiersza), umieszczany w literałach łańcuchowych Javy, które mają być użyte jako wyrażenie regularne, należy zapisać w postaci `\\n`.

---

### *java.lang.String*

#### *Opis*

Nowe metody służące do dopasowywania wyrażeń regularnych.

## Metody

`boolean matches(String wyrażenieRegularne)`

Zwraca wartość `true`, jeśli podane `wyrażenieRegularne` odpowiada całemu łańcuchowi znaków.

`String[] split(String wyrażenieRegularne)`

Zwraca tablicę łańcuchów znaków, oddzielających od siebie kolejne fragmenty łańcucha odpowiadające podanemu `wyrażeniuRegularnemu`.

`String[] split(String wyrażenieRegularne, int limit)`

Zwraca tablicę łańcuchów znaków, oddzielających od siebie pierwsze `limit-1` fragmentów łańcucha odpowiadających podanemu `wyrażeniuRegularnemu`.

`String replaceFirst(String wyrażenieRegularne, String zamiennik)`

Zastępuje podłańcuch odpowiadający podanemu `wyrażeniuRegularnemu` zamiennikiem.

`String replaceAll(String wyrażenieRegularne, String zamiennik)`

Zastępuje wszystkie podłańcuchy odpowiadające podanemu `wyrażeniuRegularnemu` zamiennikiem.

---

## *java.util.regex.Pattern*

*extends* Object

*implements* Serializable

## Opis

Reprezentuje wzorzec wyrażenia regularnego.

## Metody

`static Pattern compile(String wyrażenieRegularne)`

Tworzy nowy obiekt `Pattern` na podstawie podanego `wyrażeniaRegularnego`.

`static Pattern compile(String wyrażenieRegularne, int flagi)`  
Tworzy nowy obiekt `Pattern` na podstawie podanego *wyrażeniaRegularnego* i argumentu *flagi*, którego wartość stanowią modyfikatory trybów połączone ze sobą bitowym operatorem `OR`.

`int flags()`  
Zwraca modyfikatory trybu danego obiektu `Pattern`.

`Matcher matcher(CharSequence lancuchWejscowy)`  
Tworzy obiekt `Matcher`, który umożliwi dopasowanie tego (this) obiektu `Pattern` do podanego *łańcuchaWejscowego*.

`static boolean matches(String wyrażenieRegularne, CharSequence ↵lancuchWejscowy)`  
Zwraca wartość `true`, jeśli podane *wyrażenieRegularne* odpowiada całemu łańcuchowi przekazanemu jako *łańcuchWejscowy*.

`String pattern()`  
Zwraca wyrażenie regularne użyte do utworzenia danego obiektu `Pattern`.

`String[] split(CharSequence lancuchWejscowy)`  
Zwraca tablicę łańcuchów znaków, które rozdzielają w podanym *łańcuchuWejscowym* kolejne wystąpienia wyrażenia regularnego reprezentowanego przez dany obiekt `Pattern`.

`String[] split(CharSequence lancuchWejscowy, int limit)`  
Zwraca tablicę łańcuchów znaków, które rozdzielają w podanym *łańcuchuWejscowym* *limit-1* wystąpień wyrażenia regularnego reprezentowanego przez dany obiekt `Pattern`.

## Opis

Reprezentuje mechanizm dopasowujący wyrażenia regularne oraz wyniki dopasowania wyrażenia.

## Metody

Matcher appendReplacement(StringBuffer *sb*, String *zamiennik*)

Dołącza podłańcuch poprzedzający dopasowanie oraz *zamiennik* do bufora znakowego, określonego przy użyciu argumentu *sb*.

StringBuffer appendTail(StringBuffer *sb*)

Dołącza podłańcuch umieszczony za dopasowaniem do bufora znakowego, określonego przy użyciu argumentu *sb*.

int end()

Indeks pierwszego znaku za końcem dopasowania.

int end(int *grupa*)

Indeks pierwszego znaku znajdującego się za przechwyconą *grupą*.

boolean find()

Odnajduje kolejny fragment łańcucha pasujący do wyrażenia regularnego.

boolean find(int *poczatek*)

Odnajduje kolejny fragment łańcucha pasujący do wyrażenia regularnego, położony za znakiem o indeksie określonym jako *poczatek*.

String group()

Tekst odpowiadający wyrażeniu regularnemu reprezentowanemu przez dany obiekt Pattern.

- `String group(int grupa)`  
Tekst przechwycony przez grupę przechwytyjącą, określoną przez argument *grupa*.
- `int groupCount()`  
Liczba grup przechwytyjących zdefiniowanych w wyrażeniu reprezentowanym przez obiekt `Pattern`.
- `boolean lookingAt()`  
Zwraca `true`, jeśli dopasowanie znajduje się na samym początku wejściowego łańcucha znaków.
- `boolean matches()`  
Zwraca `true`, jeśli `Pattern` odpowiada całemu wejściowemu łańcuchowi znaków.
- `Pattern pattern()`  
Zwraca obiekt `Pattern`, wykorzystywany w danym obiekcie `Matcher`.
- `String replaceAll(String zamiennik)`  
Zastępuje każde dopasowanie podanym *zamiennikiem*.
- `String replaceFirst(String zamiennik)`  
Zastępuje pierwsze dopasowanie podanym *zamiennikiem*.
- `Matcher reset()`  
Przywraca początkowy stan mechanizmu dopasowywania, dzięki czemu kolejna operacja dopasowania rozpocznie się na samym początku wejściowego łańcucha znaków.
- `Matcher reset(CharSequence lancuchWejsciuwy)`  
Przywraca początkowy stan mechanizmu dopasowywania i określa, że ma on działać na podanym *lancuchuWejsciuwym*.
- `int start()`  
Indeks pierwszego dopasowanego znaku.

`int start(int grupa)`

Indeks pierwszego znaku dopasowanego w grupie przechwytyjącej, określonej przy użyciu argumentu *grupa*.

---

## *java.util.regex.PatternSyntaxException*

*implements* Serializable

### **Opis**

Te wyjątki są zgłaszane w celu poinformowania o pojawieniu się błędów składniowych we wzorcu wyrażenia regularnego.

### **Metody**

`PatternSyntaxException(String desc, String regex, int index)`

Tworzy nowy egzemplarz tej klasy.

`String getDescription()`

Zwraca opis błędu.

`int getIndex()`

Zwraca indeks wystąpienia błędu.

`String getMessage()`

Zwraca wielowierszowy komunikat o błędzie, zawierający jego opis, indeks, wzorzec wyrażenia regularnego oraz informacje na temat położenia miejsca wystąpienia błędu we wzorcu.

`String getPattern()`

Zwraca wzorzec wyrażenia regularnego, które zgłosiło wyjątek.

### Opis

Definiuje interfejs umożliwiający dostęp tylko do odczytu, dzięki któremu wzorce wyrażeń regularnych można zastosować do sekwencji znaków.

### Metody

char charAt(int indeks)

Zwraca znak znajdujący się w miejscu określonym przez *indeks*, przy czym indeks pierwszego znaku ma wartość 0.

int length()

Zwraca liczbę znaków w sekwencji.

CharSequence subSequence(int poczatek, int koniec)

Zwraca sekwencję zawierającą znak umieszczony w miejscu o indeksie *poczatek* i kończącą się bezpośrednio przed znakiem o indeksie *koniec*.

String toString()

Zwraca obiekt String, zawierający reprezentację danej sekwencji.