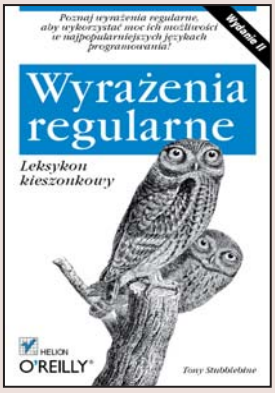
Wyrażenia regularne. Leksykon kieszonkowy. Wydanie II



Autor: Tony Stubblebine
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-1392-2
Tytuł oryginału: Regular Expression Pocket Reference
Format: B6, stron: 160

Poznaj wyrażenia regularne, aby wykorzystać moc ich możliwości w najpopularniejszych językach programowania!

- Chcesz poznać przepisy na wyrażenia regularne?
- Chcesz wykorzystywać możliwości Unicode w języku Ruby, Java, Perl, PHP, Pyton, C oraz .NET?
- Chcesz wiedzieć, jak stosować wyrażenia regularne zaimplementowane w różnych językach programowania?

Wyrażenia regularne są narzędziem umożliwiającym analizę i modyfikowanie tekstu przez dopasowywanie wzorców. Są one łańcuchem znaków zawierającym kombinację normalnych znaków oraz specjalnych metaznaków i metasekwencji, a dopasowywanie wzorców polega na odszukaniu fragmentu łańcucha opisywanego przez wyrażenie regularne. Wyrażenia te znajdują zastosowanie przy sprawdzaniu wartości zmiennych, zmianie formatu, przeprowadzaniu złożonych operacji wyszukiwania oraz weryfikowaniu poprawności danych tekstowych.

Książka „Wyrażenia regularne. Leksykon kieszonkowy" stanowi podręczny niezbędnik dla wszystkich piszących programy przetwarzające teksty. Oprócz zagadnień podstawowych, takich jak składnia wyrażeń regularnych oraz operacje, w których są wykorzystywane, leksykon zawiera inne niezwykle pomocne i bardziej zaawansowane informacje dotyczące na przykład narzędzi obsługi wyrażeń w języku Ruby oraz na serwerze WWW Apache. Czytając tę książkę, nie tylko zdobędziesz konkretną wiedzę, ale również niezbędne umiejętności praktyczne – między innymi dowiesz się, jak wykorzystać znajomość wyrażeń regularnych we wszystkich środowiskach.

- Metaznaki, tryby oraz konstrukcje
- Reprezentacja i klasy znaków
- Komentarze i modyfikatory trybów
- Narzędzia obsługi wyrażeń regularnych w języku Ruby oraz na serwerze WWW Apache
- Operatory wyrażeń regularnych w języku Perl 5.8
- Obiekty i metody do obsługi wyrażeń regularnych w języku JavaScript
- Funkcje obsługi wyrażeń regularnych w języku PHP i edytorze vi
- Obiekty i funkcje wyrażeń regularnych w języku Pyton
- Programy obsługiwane z wiersza poleceń

Wyrażenia regularne to nieocenione narzędzia w pracy programisty – nie możesz się bez nich obejść!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Podziękowania

Jeffrey Friedl napisał doskonałą książkę na temat wyrażeń regularnych — *Wyrażenia regularne* (wydaną przez wydawnictwo Helion w 2001 roku) — pisząc niniejszy *Leksykon*, korzystałem zarówno z tej książki, jak i z rad Jeffreya.

W pracach nad, jak się okazało, trudnym, pierwszym wydaniem niniejszego *Leksykonu* pomagali mi doskonali redaktorzy: Nat Torkington oraz Linda Mui. W pracach nad tym wydaniem książki wspierał mnie swym ogromnym redakcyjnym talentem Andy Oram. Na szczególne podziękowania zasługuje Sarah Burcham — za danie mi okazji do napisania niniejszej książki oraz za pomoc nad rozdziałem dotyczącym programów wykonywanych z poziomu wiersza poleceń. Podziękowania za pomoc i korektę techniczną pragnę skierować także do następujących osób: Jeffreya Friedla, Philipa Hazela, Steve'a Friedla, Ola Biniego, Iana Darwina, Zaka Greanta, Rona Hitchensa, A.M. Kuchling, Tima Allwine'a, Schuylera Erlego, Davida Lentsa, Rabble'a, Richa Bowana, Erica Eisenharta i Brada Merrilla.

Przedstawienie wyrażeń regularnych oraz zagadnień dopasowywania wzorców

Wyrażenie regularne to łańcuch znaków zawierający kombinację normalnych znaków oraz specjalnych metaznaków i metasekwencji. Normalne znaki odpowiadają samym sobie. *Metaznaki* oraz *metasekwencje* to, odpowiednio, znaki i sekwencje reprezentujące takie abstrakcje, jak ilość pewnych znaków, ich położenie lub rodzaj. Lista zamieszczona w podrozdziale „Metaznaki, tryby oraz konstrukcje" przedstawia metaznaki oraz metasekwencje najczęściej spotykane w świecie wyrażeń regularnych. W dalszych częściach książki zostały podane szczegółowe informacje dotyczące dostępności oraz składni zapisu konkretnych metaznaków w poszczególnych implementacjach wyrażeń regularnych.

Dopasowywanie wzorców polega na odszukaniu fragmentu łańcucha znaków opisywanego przez wyrażenie regularne (czyli fragmentu pasującego do tego wyrażenia). Kod, który realizuje to dopasowanie, określany jest mianem *mechanizmu wyrażeń regularnych* (ang. *Regular Expression Engine*). Wyniki wykonania większości operacji z wykorzystaniem wyrażeń regularnych można odgadnąć, pamiętając o dwóch zasadach:

1. *Pierwsze dopasowanie (położone najbardziej z lewej) wygrywa*.

   Dopasowywanie wyrażenia regularnego rozpoczyna się od pierwszego znaku łańcucha wejściowego i jest realizowane w kierunku końca tego łańcucha. Mechanizm wyrażeń regularnych kończy działanie, gdy tylko zostanie odszukany fragment łańcucha pasujący do podanego wyrażenia.

2. *Standardowe kwantyfikatory są zachłanne*.

   Kwantyfikatory określają, ile razy pewien element łańcucha znaków może być powtórzony. Standardowe kwantyfikatory starają się dopasować dany element łańcucha maksymalną liczbę razy. Element jest powtarzany mniejszą liczbę razy wyłącznie w przypadku, gdy jest to konieczne do dopasowania całości wyrażenia. Ten proces, polegający na odrzucaniu znaków i próbach mniej „zachłannego" dopasowywania, nosi nazwę sprawdzania wstecznego (ang. *backtracking*).

Pomiędzy typami poszczególnych mechanizmów obsługi wyrażeń regularnych występują subtelne różnice. Istnieją dwie klasy takich mechanizmów: Deterministyczne Automaty Skończone (w skrócie: DFA, ang. *Deterministic Finite Automaton*) oraz Niedeterministyczne Automaty Skończone (w skrócie: NFA, ang. *Nondeterministic Finite Automaton*). Mechanizmy należące do pierwszej z tych klas działają szybciej, lecz nie dysponują wieloma cechami

mechanizmów niedeterministycznych, takimi jak przechwytywanie, przewidywanie bądź też kwantyfikatory niezachłanne. Z kolei mechanizmy NFA można podzielić na dwie podklasy: tradycyjne oraz POSIX.

Mechanizmy DFA

W mechanizmach DFA każdy znak łańcucha wejściowego jest porównywany z wyrażeniem regularnym; wszystkie odnalezione dopasowania są przy tym przechowywane w pamięci. Mechanizmy tego typu są najszybsze, ponieważ każdy znak jest sprawdzany najwyżej jeden raz. W przypadku korzystania z mechanizmów DFA należy pamiętać o jednej dodatkowej zasadzie — metasekwencje alternatywy („bar|barak") są zachłanne. Gdy do wejściowego łańcucha znaków można dopasować kilka alternatywnych wersji wyrażenia regularnego, zostanie wybrana najdłuższa z nich. A zatem podaną wcześniej regułę numer 1. można by zmienić w następujący sposób: „Najdłuższe położone najbardziej z lewej strony dopasowanie wygrywa".

Tradycyjne mechanizmy NFA

W tradycyjnych mechanizmach NFA każdy element wyrażenia regularnego jest porównywany z wejściowym łańcuchem znaków; w pamięci przechowywane są miejsca, w których został dokonany wybór pomiędzy dostępnymi, alternatywnymi wersjami wyrażenia. Kiedy nie uda się dopasować danej wersji, mechanizm cofa się do ostatniej zapamiętanej pozycji. W przypadku standardowych kwantyfikatorów mechanizmy tego typu zachowują się w sposób zachłanny; jeśli jednak nie uda się w ten sposób dopasować wyrażenia, mechanizm cofa się do zapamiętanych wcześniejszych pozycji i stara się dopasować wyrażenie, postępując w sposób mniej zachłanny. W tradycyjnych mechanizmach NFA wykorzystywana jest tak zwana alternatywa uporządkowana, w której wszystkie dostępne wersje wyrażenia są sprawdzane

sekwencyjnie. Dłuższy łańcuch znaków pasujący do wyrażenia może zostać zignorowany, jeśli wcześniej uda się dopasować inną wersję wyrażenia. W tym przypadku podaną wcześniej regułę numer 1. można by zapisać w następujący sposób: „Wykorzystane zostanie pierwsze, położone najbardziej z lewej strony dopasowanie po kwantyfikatorze zachłannym".

Mechanizmy POSIX NFA
: Mechanizmy POSIX NFA działają podobnie do mechanizmów tradycyjnych NFA z jedną różnicą: mechanizmy POSIX zawsze wybierają najdłuższe położone najbardziej z lewej strony dopasowanie. Na przykład wyrażenie składające się z dwóch alternatywnych wersji `kat|kategoria` zawsze zwróci słowo „kategoria", o ile oczywiście jest to możliwe, nawet jeśli w pierwszej kolejności uda się dopasować pierwszą wersję wyrażenia (`kat`).

Metaznaki, tryby oraz konstrukcje

Metaznaki oraz metasekwencje przedstawione w tej części rozdziału stanowią najczęściej spotykane typy konstrukcji używanych przy tworzeniu wyrażeń regularnych. Jednocześnie podana została także najczęściej spotykana składnia zapisu tych metaznaków i metasekwencji. Należy jednak pamiętać, że zarówno sposób zapisu, jak i możliwości jego stosowania są różne w różnych implementacjach wyrażeń regularnych.

Reprezentacja znaków

W wielu implementacjach wyrażeń regularnych dostępne są skróty reprezentujące znaki, które w innych przypadkach trudno by było zapisać.

Skrócone reprezentacje znaków

W większości implementacji wyrażeń regularnych dostępne są skróty reprezentujące następujące znaki: `alarm`, `backspace`, `escape`, `przesunięcie wiersza`, `nowy wiersz`, `powrót karetki`, `poziomy znak tabulacji` oraz `pionowy znak tabulacji`. Na przykład `\n` jest często stosowanym skrótowym sposobem zapisu znaku nowego wiersza, którym zazwyczaj jest znak `LF` (012 ósemkowo), lecz czasami — w zależności od używanego systemu operacyjnego — może nim także być znak `CR` (015 ósemkowo). Sporych problemów może także przysporzyć skrót `\b`, który w wielu implementacjach wyrażeń regularnych oznacza jednocześnie znak cofnięcia (`backspace`), jak również granicę słowa (czyli miejsce, gdzie znak będący częścią słowa graniczy ze znakiem, który nie jest częścią słowa). W takich przypadkach w klasie znaków (czyli zbiorze znaków, których można użyć do dopasowania) `\b` oznacza znak cofnięcia, a we wszystkich pozostałych przypadkach — granicę słowa.

Ósemkowy zapis unikowy: `\liczba`

Reprezentuje znak odpowiadający dwu- lub trzycyfrowej liczbie ósemkowej. Na przykład `\015\012` odpowiada sekwencji znaków ASCII `CR LF`.

Szesnastkowy zapis unikowy oraz zapis Unicode: `\xliczba`, `\x{liczba}`, `\uliczba`, `\Uliczba`

Reprezentuje znak odpowiadający liczbie szesnastkowej. Czterocyfrowe lub większe liczby szesnastkowe mogą reprezentować zakresy znaków Unicode. Na przykład `\x0D\x0A` odpowiada sekwencji znaków ASCII `CR LF`.

Znaki sterujące: `\cznak`

Odpowiadają one znakom sterującym kodu ASCII reprezentowanym przez wartości mniejsze od 32. Dla bezpieczeństwa *znak* zawsze należy zapisywać wielkimi literami,

gdyż niektóre implementacje nie pozwalają w tych przypadkach na stosowanie małych liter. Na przykład `\cH` odpowiada kombinacji *Control-H*, czyli znakowi cofnięcia w kodzie ASCII.

Klasy znaków oraz skrótowe zapisy klas

Klasy znaków to sposób definiowania lub określania zbiorów znaków. Klasa znaków odpowiada jednemu znakowi łańcucha wejściowego, który należy do zdefiniowanego zbioru znaków.

Normalne klasy: `[...]` *oraz* `[^...]`
: Klasy znaków — `[...]` — oraz zanegowane klasy znaków — `[^...]` — pozwalają na określenie znaków, które mogą oraz które nie mogą być dopasowywane. Klasa znaków zawsze odpowiada jednemu znakowi łańcucha wejściowego. Znak `-` (minus) oznacza pewien zakres znaków. Na przykład wyrażenie `[a-z]` odpowiada dowolnej małej literze kodu ASCII. Aby umieścić sam minus (`-`) w klasie znaków, należy go poprzedzić znakiem unikowym.

Prawie każdy znak: kropka (`.`)
: Zazwyczaj kropka odpowiada dowolnemu znakowi za wyjątkiem znaku nowego wiersza. Często jednak można zmienić tryb działania dopasowywania w taki sposób, aby kropka odpowiadała także znakom nowego wiersza. W obrębie klasy znaków kropka zawsze odpowiada kropce.

Skrótowe zapisy klas znaków: `\w`, `\d`, `\s`, `\W`, `\D`, `\S`
: Są to często spotykane skrótowe sposoby zapisu klas znaków reprezentujących słowa, cyfry oraz znaki odstępu. Najczęściej za *znaki mogące tworzyć słowa* uznawane są wszystkie znaki alfanumeryczne kodu ASCII oraz znak podkreślenia. Niemniej jednak, w zależności od implementacji wyrażeń regularnych, do grupy znaków alfanumerycznych mogą być także zaliczane znaki lokalne oraz alfanumeryczne znaki Unicode. Skrótowy zapis, w którym wykorzystano małą literę

(na przykład `/s`), odpowiada jednemu znakowi z danej klasy; natomiast zapis, w którym zastosowano wielką literę (na przykład `/S`), odpowiada znakowi, który do danej klasy nie należy. Na przykład `\d` odpowiada pojedynczej cyfrze i zazwyczaj znaczy to samo co wyrażenie `[0-9]`.

Klasy znaków POSIX: `[:klasa:]`
: Standard POSIX definiuje kilka klas znaków, których można używać wyłącznie w klasach znaków wyrażeń regularnych (patrz tabela 1). Na przykład przyjrzyjmy się klasie `[:lower:]`. Jeśli zostanie ona zapisana jako `[[:lower:]]`, to będzie odpowiadać wyrażeniu `[a-z]` kodu ASCII dla danych ustawień lokalnych.

Tabela 1. Klasy znaków POSIX

Klasa	Znaczenie
`alnum`	Litery i znaki
`alpha`	Litery
`blank`	Jedynie odstęp lub znak tabulacji
`cntrl`	Znaki sterujące
`digit`	Cyfry (dziesiętnego systemu liczbowego)
`graph`	Znaki drukowalne bez odstępu
`lower`	Małe litery
`print`	Znaki drukowalne wraz ze znakiem odstępu
`punct`	Znaki drukowalne z wyłączeniem liter i cyfr
`space`	Odstępy
`upper`	Duże litery
`xdigit`	Cyfry szesnastkowe

Właściwości Unicode, pisma oraz bloki:
`\p{właściwość}, \P{właściwość}`
: Standard Unicode definiuje klasy znaków, które mają szczególne właściwości, należą do pewnego pisma (ang. *script*) lub istnieją w pewnym bloku. *Właściwości* określają na przykład, czy dany znak jest literą, czy też cyfrą (patrz tabela 2).

Pismo określa różne systemy piśmiennicze, na przykład hebrajski, łaciński czy też Han. Z kolei *bloki* to zakresy znaków na mapie znaków Unicode. W niektórych implementacjach wyrażeń regularnych właściwości Unicode muszą być poprzedzane łańcuchami `Is` lub `In`. Na przykład `\p{Ll}` odpowiada dowolnej małej literze w dowolnym języku uwzględnionym w Unicode, czyli przykładowo literze `a` lub α.

Tabela 2. Standardowe właściwości Unicode

Właściwość	Znaczenie
`\p{L}`	Litery
`\p{Ll}`	Małe litery
`\p{Lm}`	Litery modyfikatorów
`\p{Lo}`	Inne litery. Nie mają one wielkości ani nie są uznawane za modyfikatory
`\p{Lt}`	Litery tytułowe
`\p{Lu}`	Wielkie litery
`\p{C}`	Kody sterujące oraz znaki, które nie należą do innych kategorii
`\p{Cc}`	Znaki sterujące kodów ASCII oraz Latin-1
`\p{Cf}`	Niewidoczne znaki formatujące
`\p{Cn}`	Nieprzypisane punkty kodu
`\p{Co}`	Do zastosowań prywatnych, na przykład logo firmy
`\p{Cs}`	Znaki zastępujące
`\p{M}`	Znaczniki przeznaczone do łączenia ze znakami bazowymi, na przykład akcenty
`\p{Mc}`	Znaki modyfikujące, wymagające odrębnego miejsca. Przykładem mogą tu być „znaki samogłosek”
`\p{Me}`	Znaki otaczające inne znaki, na przykład okręgi, kwadraty oraz romby
`\p{Mn}`	Znaki modyfikujące inne znaki, na przykład akcenty bądź też znaki przegłosu
`\p{N}`	Znaki liczbowe
`\p{Nd}`	Liczby dziesiętne w różnych pismach
`\p{Nl}`	Litery będące cyframi, na przykład cyfry rzymskie

Tabela 2. Standardowe właściwości Unicode (ciąg dalszy)

Właściwość	Znaczenie
`\p{No}`	Indeksy górne i dolne, symbole oraz inne znaki reprezentujące liczby, które jednak nie są cyframi
`\p{P}`	Znaki przestankowe
`\p{Pc}`	Łączące znaki przestankowe, na przykład znak podkreślenia
`\p{Pd}`	Kreski i łączniki
`\p{Pe}`	Zamykające znaki przestankowe, dopełniające znaki reprezentowane przez `\p{Ps}`
`\p{Pi}`	Początkowe znaki przestankowe, takie jak cudzysłów otwierający
`\p{Pf}`	Końcowe znaki przestankowe, takie jak cudzysłów zamykający
`\p{Po}`	Inne znaki przestankowe
`\p{Ps}`	Otwierające znaki przestankowe, na przykład nawias otwierający
`\p{S}`	Symbole
`\p{Sc}`	Waluty
`\p{Sk}`	Znaki łączące, reprezentowane jako niezależne znaki
`\p{Sm}`	Symbole matematyczne
`\p{So}`	Inne symbole
`\p{Z}`	Znaki oddzielające, które nie posiadają żadnej widocznej reprezentacji
`\p{Zl}`	Separatory wierszy
`\p{Zp}`	Separatory akapitów
`\p{Zs}`	Znaki odstępu

Sekwencja znaków łączonych Unicode: `\X`

Odpowiada znakowi bazowemu Unicode, po którym może być umieszczona dowolna liczba znaków łączących Unicode. Jest to uproszczony sposób zapisu wyrażenia `\P{M}\p{M}`. Na przykład `\X` odpowiada znakowi è, jak również dwóm znakom `e'`.

Punkty zakotwiczenia oraz warunki o zerowej długości

Punkty zakotwiczenia oraz warunki o zerowej długości odpowiadają miejscom w wejściowym łańcuchu znaków.

Początek wiersza lub łańcucha znaków: `^`, `\A`

Pasuje do początku przeszukiwanego łańcucha znaków. W trybie wielowierszowym `^` pasuje do każdego miejsca bezpośrednio po dowolnym znaku nowego wiersza. Niektóre implementacje wyrażeń regularnych udostępniają metaznak `\A`, który odpowiada samemu początkowi całego tekstu.

Koniec wiersza lub łańcucha znaków: `$`, `\Z`, `\z`

Metaznak `$` pasuje do końca łańcucha znaków. W sytuacjach, gdy dostępny jest metaznak `\Z`, pasuje on do końca łańcucha lub miejsca położonego bezpośrednio przed znakiem końca wiersza kończącego łańcuch znaków — i to niezależnie od trybu, w jakim jest przeprowadzane dopasowywanie. Niektóre implementacje wyrażeń regularnych obsługują także metaznak `\z`, który — niezależnie od trybu dopasowywania — pasuje do samego końca łańcucha.

Początek dopasowywania: `\G`

W przypadku dopasowywania interaktywnego metaznak `\G` pasuje do miejsca, w którym zostało zakończone poprzednie dopasowywanie. Często zdarza się, że w przypadku nieudanego dopasowywania miejsce to zostaje przeniesione na sam początek przeszukiwanego łańcucha znaków.

Granica słowa: `\b`, `\B`, `\<`, `\>`

Metaznaki reprezentujące granicę słowa pasują do miejsca, w którym znak mogący tworzyć słowa sąsiaduje ze znakiem, który nie może tworzyć słowa. Bardzo często metaznak `\b` określa miejsce, w którym znajduje się granica słowa, a metaznak `\B` — miejsce, w którym granicy słowa nie ma.

Niektóre implementacje wyrażeń regularnych udostępniają także niezależne metaznaki określające granice początku i końca słowa; często są to `\<` oraz `\>`.

Przewidywanie: `(?=...)`, `(?!...)`
Przewidywanie wsteczne: `(?<=...)`, `(?<!...)`
: *Konstrukcje przewidywań* pasują odpowiednio do tych miejsc w tekście, gdzie zostałby dopasowany podwzorzec (przewidywanie), gdzie podwzorzec nie zostałby dopasowany (przewidywanie negatywne), gdzie znalazłby się koniec dopasowanego podwzorca (przewidywanie wsteczne) lub gdzie nie znalazłby się koniec dopasowanego podwzorca (negatywne przewidywanie wsteczne). Na przykład `pi(?=wo)` pasuje do liter `pi` w słowie `piwo`, ale nie w słowie `pika`. W niektórych implementacjach wyrażeń regularnych konstrukcje przewidywania wstecznego są ograniczane do podwzorców o z góry określonej długości.

Komentarze i modyfikatory trybów

Modyfikatory trybu pozwalają na zmienianie sposobu, w jaki mechanizmy wyrażeń regularnych interpretują wyrażenia regularne.

Tryb wielowierszowy: `m`
: Zmienia działanie metaznaków `^` oraz `$` w taki sposób, iż pasują one do miejsc sąsiadujących ze znakami nowego wiersza znajdującymi się w wejściowym łańcuchu znaków.

Tryb jednowierszowy: `s`
: Zmienia działanie metaznaku `.` (kropka) w taki sposób, iż odpowiada on wszystkim znakom łańcucha wejściowego, w tym także znakom nowego wiersza.

Tryb ignorowania wielkości liter: `i`
: Sprawia, że litery różniące się wyłącznie wielkością są uważane za identyczne.

Tryb ignorowania odstępów: `x`

W tym trybie w wyrażeniach regularnych można umieszczać dowolne ilości odstępów oraz komentarze. W tym trybie mechanizmy obsługi wyrażeń regularnych ignorują wszystkie białe znaki oraz komentarze (rozpoczynające się od znaku # i rozciągające się aż do końca wiersza).

Modyfikatory trybu: `(?i)`, `(?-i)`, `(?`*`tryb`*`:...)`

Zazwyczaj modyfikatory trybu można umieszczać bezpośrednio w wyrażeniach regularnych — `(?`*`tryb`*`)` włącza wybrany tryb w dalszej części podwyrażenia, `(?-`*`tryb`*`)` wyłącza podany tryb, a `(?`*`tryb`*`:...)` włącza lub wyłącza podany tryb w wyrażeniu podanym między dwukropkiem i nawiasem zamykającym. Na przykład `używaj (?i:perla)` pasuje do łańcuchów `używaj Perla`, `używaj PerLA` i tak dalej.

Komentarze: `(?#...)` *oraz* `#`

W trybie ignorowania odstępów znak # informuje, że pozostała część wiersza stanowi komentarz. Jeśli konstrukcja ta jest obsługiwana, to obszar komentarza — `(?#...)` — można umieszczać w dowolnym miejscu wyrażenia regularnego i to niezależnie od używanego trybu. Na przykład w wyrażeniu `.{0,80}(?#Limit długości wynosi 80 znaków)` można było umieścić komentarz, dlaczego użyto `{0,80}`.

Obszar zapisu dosłownego: `\Q...\E`

W tym przypadku wszystkie metaznaki zapisane pomiędzy `\Q` oraz `\E` są traktowane w sposób dosłowny. Na przykład wyrażenie `\Q(.*)\E` ma to samo znaczenie co wyrażenie `\(\.\*\)`.

Grupowanie, przechwytywanie, warunki i sterowanie

W tej części rozdziału została opisana składnia pozwalająca na grupowanie podwzorców, przechwytywanie dopasowanych fragmentów wyrażenia, warunkowe dopasowywanie fragmentów wyrażenia oraz określanie, ile razy może pojawić się określony podwzorzec.

Nawiasy przechwytujące i grupujące: `(...)` *oraz* `\1`, `\2`, *...*

Nawiasy spełniają w wyrażeniach regularnych dwie funkcje: grupują oraz przechwytują. Tekst pasujący do podwyrażenia zapisanego w nawiasach zostaje „przechwycony" i zapamiętany, dzięki czemu można z niego korzystać w dalszej części wyrażenia. Pary nawiasów przechwytujących są numerowane na podstawie kolejności, w jakiej są zapisane ich nawiasy otwierające, zaczynając od lewej strony. Jeśli mechanizm obsługi wyrażeń regularnych obsługuje odwołania wsteczne, to podczas dopasowywania wyrażenia można się w nim odwoływać do wcześniej dopasowanych podwyrażeń; do tego celu służą symbole `\1`, `\2` i tak dalej. Przechwycone fragmenty tekstu są też dostępne po zakończeniu dopasowywania wyrażenia regularnego, jednak sposoby ich odczytu zależą od konkretnej implementacji wyrażeń regularnych. Na przykład wyrażenie `\b(\w+)\b\s+\1\b` pasuje do powtórzonych słów, takich jak `hej hej`.

Nawiasy jedynie grupujące: `(?:...)`

Ten rodzaj nawiasów jedynie grupuje podwyrażenie, które następnie może być wykorzystane w jakimś celu, lecz nie powoduje przechwycenia tekstu. Rozwiązanie to jest przydatne ze względu na efektywność i możliwość wielokrotnego wykorzystania. Na przykład `(?:futro)` odpowiada słowu `futro`, lecz nie powoduje jego zapamiętania.

Nazwane przechwycenia: `(?<nazwa>...)`

Konstrukcja ta realizuje grupowanie i przechwycenie tekstu pasującego do podanego podwyrażenia, a jednocześnie pozwala na późniejsze odwoływanie się do tego tekstu przy użyciu *nazwy*. Na przykład wyrażenie `Temat:(?<temat>.*)` zapisuje łańcuch znaków podany po słowie `Temat:` do grupy, do której można się odwoływać przy użyciu nazwy `temat`.

Grupowanie atomowe: `(?>...)`

Sprawia, że mechanizm wyrażeń regularnych nigdy nie wykorzysta ponownie podwyrażenia podanego w grupie, nawet jeśli miałoby to sprawić, że nie uda się dopasować całego wyrażenia regularnego. Na przykład wyrażenie `(?>[ab]*)\w\w` będzie pasować do łańcucha znaków `aabbcc`, lecz nie do `aabbaa`.

Alternatywa: `...|...`

Pozwala na sprawdzenie kilku podwyrażeń. Niski priorytet alternatywy sprawia, że podwyrażenia są dłuższe, niż zamierzano; dlatego też podwyrażenia wchodzące w skład alternatywy należy zapisywać wewnątrz nawiasów. Na przykład wyrażenie `\b(ptak|smak)\b` pasuje do wyrazu `ptak` lub wyrazu `smak`.

Wyrażenie warunkowe: `(?(`*jeżeli*`) to | `*w_przeciwnym_razie*`)`

Postać części *jeżeli* jest zależna od implementacji wyrażeń regularnych, jednak zazwyczaj jest to odwołanie do przechwyconego podwyrażenia lub przewidywanie. Z kolei części *to* oraz *w_przeciwnym_razie* są wzorcami wyrażenia regularnego. Jeśli warunek określony przez część *jeżeli* zostanie sprawdzony, zostanie zastosowane wyrażenie *to*, a w przeciwnym przypadku — wyrażenie *w_przeciwnym_razie*. Na przykład wyrażenie `(<)?bom(?(1)>|ba)` pasuje do łańcucha `<bom>` i `bomba`.

Kwantyfikatory zachłanne: `*`, `+`, `?`, `{liczba, liczba}`
: Kwantyfikatory zachłanne określają, ile razy może zostać zastosowana dana konstrukcja. Dana konstrukcja jest zazwyczaj wykorzystywana maksymalną liczbę razy, jednak kwantyfikatory mogą także ograniczyć liczbę jej powtórzeń, jeśli dzięki temu uda się dopasować całe wyrażenie regularne. Na przykład wyrażenie `(ab)+` pasuje do łańcucha `abababababab` składającego się z dowolnej liczby powtórzeń łańcucha `ab`.

Kwantyfikatory leniwe: `*?`, `+?`, `??`, `{liczba, liczba}?`
: Kwantyfikatory tego typu określają, ile razy dana konstrukcja może zostać zastosowana. Niemniej jednak, w odróżnieniu od kwantyfikatorów zachłannych, konstrukcje są najczęściej dopasowywane minimalną liczbę razy. Na przykład wyrażenie `(an)+?` pasuje jedynie do fragmentu `an` słowa `banana`.

Kwantyfikatory własnościowe: `*+`, `++`, `?+`, `{liczba, liczba}+`
: Kwantyfikatory własnościowe działają podobnie jak kwantyfikatory zachłanne, z tą różnicą, iż „blokują" fragmenty łańcucha wejściowego, które udało im się dopasować, i nie pozwalają na późniejsze cofanie się i dzielenie tych fragmentów. Na przykład wyrażenie `(ab)++ab` nie będzie pasować do łańcucha znaków `abababab`.

Obsługa Unicode

Unicode to zbiór znaków, w którym wszystkim znakom występującym we wszystkich językach świata przypisane są unikalne numery. Ze względu na dużą ilość występujących znaków w Unicode każdy znak musi być reprezentowany przez więcej niż jeden bajt. Niektóre implementacje wyrażeń regularnych nie rozpoznają znaków Unicode, gdyż oczekują jednobajtowych znaków kodu ASCII. Podstawowa obsługa Unicode może zatem polegać na

samej możliwości dopasowania wyrażenia składającego się ze znaków Unicode. Bardziej zaawansowana obsługa może obejmować rozpoznawanie klas znaków oraz innych konstrukcji zawierających znaki ze wszystkich języków uwzględnianych w Unicode. Na przykład `\w` może odpowiadać zarówno literze `è`, jak i `e`.

Przepisy na wyrażenia regularne

W tej części rozdziału zamieszczone zostały proste przykłady wzorców wyrażeń regularnych. Być może konieczne będzie wprowadzenie w nich pewnych modyfikacji, by dostosować je do konkretnych potrzeb.

Wraz z każdym wyrażeniem prezentowane są także łańcuchy znaków, które do niego pasują, oraz takie, które nie pasują. Dzięki temu Czytelnikowi łatwiej będzie określić, jakie zmiany należy wprowadzić w wyrażeniu, by dostosować je do własnych potrzeb.

Wyrażenia są zapisane w sposób charakterystyczny dla języka Perl:

```
/wzorzec/tryb
s/wzorzec/zamiennik/tryb
```

Przepisy

Usuwanie odstępów na początku i końcu łańcucha

```
s/^\s+//
s/\s+$//
```

Pasuje do: `"  bar kod   ", "bar    "`

Nie pasuje do: `"bar kod"`