# Lekcja - wyrażenia regularne

Wyobraź sobie, że masz duży tekstowy plik, a twoim zadaniem jest znalezienie wszystkich słów, które zaczynają się od litery "A". Wyrażenia regularne to narzędzie, które pozwala ci to zrobić w sposób bardziej zaawansowany niż ręczne przeszukiwanie tekstu.

Przykładowo, jeśli masz taki tekst:

"Aplikacja Android jest bardzo popularna, a Android to system operacyjny."

Możesz użyć wyrażenia regularnego "A\w+" (gdzie "\w+" oznacza dowolny ciąg liter) do znalezienia wszystkich słów zaczynających się od "A". W wyniku uzyskasz:

"Aplikacja", "Android", "a", "Android"

Wyrażenia regularne pozwalają na bardziej zaawansowane operacje, takie jak szukanie wszystkich adresów email w tekście, numerów telefonów, dat, czy innych wzorców. Są używane w programowaniu, przetwarzaniu tekstu, analizie danych i wielu innych dziedzinach informatyki. Pomagają automatyzować procesy wyszukiwania i analizy tekstu.

Wyrażenia regularne to potężne narzędzie używane w programowaniu i przetwarzaniu danych do analizy i manipulacji tekstami. Pozwalają nam znaleźć i wyciągnąć z tekstu konkretne informacje, takie jak adresy email, numery telefonów, daty, linki internetowe czy jakiekolwiek inne wzorce znaków. Dzięki nim możemy szybko i efektywnie przeszukiwać duże zbiory tekstów, wykonywać operacje takie jak zastępowanie, usuwanie lub zamianę ciągów znaków, walidować dane wejściowe (na przykład sprawdzając, czy wprowadzony numer telefonu ma właściwy format), a także parsować i analizować struktury tekstowe, takie jak pliki konfiguracyjne, logi czy strony internetowe. Wyrażenia regularne są więc nieocenionym narzędziem w pracy programistów, administratorów systemów, analityków danych i wszystkich tych, którzy muszą pracować z tekstami w sposób bardziej zaawansowany niż prosty tekstowy wyszukiwacz.

## Wyrażenia regularne - historia

Historia wyrażeń regularnych sięga lat 40. XX wieku, kiedy to matematyk Stephen Kleene opracował formalizm matematyczny nazwany "automatami skończonymi", który był bazą dla wyrażeń regularnych. Automaty skończone były wykorzystywane do opisywania wzorców i sekwencji znaków w tekstach oraz do rozwiązywania problemów związanych z przetwarzaniem ciągów znaków. Termin "wyrażenia regularne" stał się popularny w latach 50. i 60., a teoria ta znalazła zastosowanie w informatyce i informatyce teoretycznej.

Podczas rozwoju komputerów i języków programowania, wyrażenia regularne stały się częścią narzędzi do przetwarzania tekstu. Są obsługiwane przez wiele języków programowania, takich jak Perl, Python, JavaScript, Ruby, Java, C# i wiele innych. Każdy z tych języków ma własną implementację wyrażeń regularnych, ale ogólny zarys ich działania jest podobny. Istnieją pewne różnice w składni i funkcjonalności, ale większość podstawowych koncepcji pozostaje taka sama.

Warto zaznaczyć, że choć podstawowe zasady wyrażeń regularnych są podobne, to składnia i dostępne funkcje mogą się różnić między językami programowania.

## Składnia wyrażeń regularnych

## Anchors

Znaki anchorów (kotwic) w wyrażeniach regularnych służą do określenia konkretnych pozycji lub lokalizacji w tekście, gdzie ma nastąpić dopasowanie wzorca.

^ - Oznacza dopasowanie do początku linii. Na przykład, wyrażenie "^r" dopasuje słowo, które zaczyna się od litery "r".

$ - Oznacza dopasowanie do końca linii. Na przykład, wyrażenie "t$" dopasuje słowo, które kończy się literą "t".

\A - Oznacza dopasowanie do początku linii. Jest podobne do "^", ale jest bardziej ogólnym znakiem specjalnym, który działa nie tylko w kontekście całego tekstu, ale także w wieloliniowych ciągach znaków.

\Z - Oznacza dopasowanie do końca linii. Jest podobne do "$", ale działa również w kontekście wieloliniowych ciągów znaków.

\b - Oznacza dopasowanie do początku lub końca słowa (czyli ciągu znaków oddzielonych od innych białymi znakami lub znakami nie będącymi literami i cyframi). Na przykład, "\bfox\b" dopasuje słowo "fox" tylko wtedy, gdy jest oddzielone od innych znaków.

\B - Oznacza dopasowanie do znaków znajdujących się w środku słowa, czyli tam, gdzie nie ma białych znaków ani innych znaków niebędących literami i cyframi. Na przykład, "\Bee\B" dopasuje słowo "bee" tylko wtedy, gdy jest otoczone innymi literami lub cyframi.

## Znaki specjalne

"." (Kropka):

Ten znak pasuje do dowolnego znaku, z wyjątkiem znaku nowej linii (linebreak).

Na przykład, wyrażenie "c.e" dopasuje "cat", "cze", "c4e" itd., ponieważ kropka pasuje do dowolnego znaku między literami "c" i "e".

"\d" (Cyfra):

Ten znak pasuje do dowolnej cyfry od 0 do 9.

Na przykład, "\d" dopasuje dowolną cyfrę, taką jak 0, 5 czy 9.

"\D" (Nie-cyfra):

Ten znak pasuje do dowolnego znaku, który nie jest cyfrą.

Na przykład, "\D" dopasuje literę, spację, znak specjalny lub cokolwiek innego, co nie jest cyfrą.

"\w" (Litera lub cyfra):

Ten znak pasuje do dowolnej litery lub cyfry lub znaku podkreślenia "\_".

Na przykład, "\w" dopasuje litery "a", "B", cyfry "7", lub znak podkreślenia "\_".

"\W" (Nie-litera, nie-cyfra, nie-\_):

Ten znak pasuje do dowolnego znaku, który nie jest literą, cyfrą ani znakiem podkreślenia

"\_".

Na przykład, "\W" dopasuje spację, znaki specjalne, kropki i inne znaki niebędące literami, cyframi lub "\_".

"\s" (Biały znak):

Ten znak pasuje do dowolnego białego znaku, takiego jak spacja, tabulator lub znak nowej linii.

Na przykład, "\s" dopasuje spację, tabulator lub znak nowej linii.

"\S" (Nie-biały znak):

Ten znak pasuje do dowolnego znaku, który nie jest białym znakiem, czyli np. litera, cyfra lub znak specjalny.

Na przykład, "\S" dopasuje litery, cyfry, znaki interpunkcyjne itd.

"\metaznak" (Unikanie metaznaku):

Jeśli chcesz dopasować dokładnie znak specjalny, który normalnie jest traktowany jako metaznak (np. ".", "^"), musisz go "wyciągnąć" z metaznaku za pomocą ukośnika "".

Na przykład, aby dopasować kropkę, używamy ".", aby dopasować znak "^", używamy "^".

## Character classes

Character classes (klasy znaków) w wyrażeniach regularnych to zestawy lub zakresy znaków, które pozwalają na dopasowanie jednego z kilku określonych znaków w danym miejscu w tekście. Oto opis wyrażeń związanych z klasami znaków:

[xy] (Klasa znaków):

To wyrażenie regularne pasuje do jednego z określonych znaków, które znajdują się w nawiasach kwadratowych.

Na przykład, wyrażenie gr[ea]y dopasuje zarówno "grey", jak i "gray", ponieważ pasuje do "e" lub "a" między literami "g" i "y".

[x-y] (Zakres znaków):

To wyrażenie regularne pasuje do znaków znajdujących się w określonym zakresie między dwiema literami w nawiasach kwadratowych.

Na przykład, [a-e] dopasuje dowolny znak z zakresu od "a" do "e", więc pasuje do liter "a", "b", "c" lub "e".

[^xy] (Negacja klasy znaków):

To wyrażenie regularne pasuje do znaków, które nie są wymienione w nawiasach kwadratowych.

Na przykład, gr[^ea]y dopasuje "grey", ale nie dopasuje "gray", ponieważ nie pasuje do "e" ani "a" między "g" i "y".

[\^-] (Znak metaznaku w klasie znaków):

Jeśli chcemy dopasować znak, który normalnie jest traktowany jako metaznak, wewnątrz klasy znaków, musimy go unikać, dodając ukośnik "" przed nim.

Na przykład, 4[\^\.-+\*/]\d dopasuje sekwencję, która zaczyna się od "4", a następnie ma dowolny znak metaznaku (np. "^", ".", "-", "+", "\*", "/") i na końcu cyfrę.

## Repetition

Znaki Repetition w wyrażeniach regularnych pozwalają na dopasowanie powtarzających się znaków lub sekwencji znaków w tekście. Oto opis przykładów związanym z tymi znakami:

x\* (Zero lub więcej razy):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" powtarzany zero lub więcej razy.

Na przykład, wyrażenie ar\*o dopasuje słowa "ao", "aro", "aarro" i inne, ponieważ "r" po "a" może występować zero lub więcej razy.

x+ (Jeden lub więcej razy):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" powtarzany co najmniej raz.

Na przykład, wyrażenie re+ dopasuje słowa "re", "ree", "reee" i tak dalej, ponieważ "e" po "r" musi występować co najmniej raz.

x? (Zero lub jeden raz):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" zero lub dokładnie jeden raz.

Na przykład, wyrażenie ro?a dopasuje słowa "roa" i "ra", ponieważ "o" może występować zero lub jeden raz po "r".

x{m} (Dokładnie m razy):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" dokładnie "m" razy.

Na przykład, \we{2}\w dopasuje słowo, które zawiera dwie litery "e" obok siebie, na przykład "geese" (gęsi).

x{m,} (M razy lub więcej):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" co najmniej "m" razy.

Na przykład, 2{3,}4 dopasuje "2224", "22224", "222224" i tak dalej, ponieważ "2" musi występować co najmniej trzy razy.

x{m,n} (Od m do n razy):

To wyrażenie regularne pasuje do sekwencji, która zawiera znak "x" od "m" do "n" razy (włącznie).

Na przykład, 12{1,3}3 dopasuje "123", "1223" i "12223", ponieważ "2" może występować od jednego do trzech razy.

x\*?, x+?, itp. (Leniwe ilości):

Te znaki są dodawane po znakach repetycji (na przykład \*?, +?) i oznaczają dopasowanie minimalnej ilości znaków.

Domyślnie wyrażenia regularne dopasowują jak najwięcej znaków, ale dodanie ? sprawi, że będą dopasowane jak najmniejsza ilość znaków. To jest znane jako "lazy quantifier".

Na przykład, re+? dopasuje "re", a nie "ree" czy "reee", ponieważ jest leniwe i stara się dopasować jak najmniejszą ilość znaków.

## Capturing groups

Capturing groups (grupy przechwytujące) w wyrażeniach regularnych pozwalają na wyodrębnienie i zapisanie konkretnych części tekstu z dopasowanego wzorca. Poniżej opisuję przykłady związanego z nimi:

(x) (Grupa przechwytująca):

To wyrażenie regularne umieszczone w nawiasach tworzy grupę przechwytującą. Pozwala ono na zachowanie i wyodrębnienie tekstu pasującego do wzorca w danej grupie.

Na przykład, (iss)+ dopasuje i zachowa sekwencje "iss" w tekście, a jeśli będzie ich więcej, to zostaną one również zachowane.

(?:x) (Grupa bez przechwytywania):

To jest podobne do grupy przechwytującej, ale nie zapisuje dopasowanego tekstu w grupie. Jest to przydatne, gdy chcemy grupować wzorce bez przechwytywania.

Na przykład, (?:ab)(cd) dopasuje "abcd", ale tylko "cd" zostanie zachowane w grupie.

(?<name>x) (Nazwana grupa przechwytująca):

To jest rozszerzenie grupy przechwytującej, które pozwala nazwać grupę, aby łatwiej odwoływać się do jej zawartości.

Na przykład, (?<first>\d)(?<second>\d)\d\* dopasuje ciąg cyfr i zapisze pierwszą cyfrę w grupie o nazwie "first", a drugą cyfrę w grupie o nazwie "second".

(x|y) (Alternatywne wzorce):

To wyrażenie regularne pozwala na dopasowanie jednego z kilku alternatywnych wzorców. Możemy użyć nawiasów, aby ograniczyć zakres alternatywnych wzorców.

Na przykład, (re|ba) dopasuje "re" lub "ba", pasując do dowolnego z tych dwóch wzorców.

\n (Referencja do poprzednich grup przechwytujących):

Pozwala na odwołanie się do zawartości wcześniej zdefiniowanych grup przechwytujących w wyrażeniu regularnym.

Na przykład, (b)(\w\*)\1 dopasuje słowo, które zaczyna się od litery "b" i potem ma dowolny ciąg liter, ale zawsze zakończone jest tą samą literą "b", która została dopasowana wcześniej.

\k<name> (Referencja do nazwanej grupy przechwytującej):

Pozwala na odwołanie się do zawartości wcześniej zdefiniowanej nazwanej grupy przechwytującej w wyrażeniu regularnym.

Na przykład, (?<first>5)(\d\*)\k<first> dopasuje ciąg, który zaczyna się od cyfry "5", a następnie ma dowolny ciąg cyfr, ale zawsze jest kończony tą samą cyfrą "5", która jest zdefiniowana w nazwanej grupie "first".

## Lookahead

Operatory lookahead (z wyrażeniem regularnym w nawiasach) pozwalają na sprawdzenie, czy określone znaki lub wzorce znajdują się przed lub po dopasowaniu wzorca, bez uwzględniania tych znaków w samym dopasowaniu. Oto opis tych operatorów wraz z przykładami:

(?=x) (Positive Lookahead):

Ten operator lookahead sprawdza, czy znaki lub wzorzec "x" występują bezpośrednio po dopasowaniu danego wzorca, ale nie włącza ich w dopasowanie.

Na przykład, an(?=an) dopasuje "an", ale tylko wtedy, gdy po nim występuje kolejne "an", np. dopasuje "anan", ale nie dopasuje "and" ani "angel".

(?!x) (Negative Lookahead):

Ten operator lookahead sprawdza, czy znaki lub wzorzec "x" nie występują bezpośrednio po dopasowaniu danego wzorca.

Na przykład, ai(?!n) dopasuje "ai", ale tylko jeśli nie występuje po nim "n". Dopasuje "air", ale nie dopasuje "ain".

(?<=x) (Positive Lookbehind):

Ten operator lookahead sprawdza, czy znaki lub wzorzec "x" występują bezpośrednio przed dopasowaniem danego wzorca, ale nie włącza ich w dopasowanie.

Na przykład, (?<=tr)a dopasuje "a", ale tylko wtedy, gdy przed nim występuje "tr", np. dopasuje "tra", ale nie dopasuje "ara".

(?<!x) (Negative Lookbehind):

Ten operator lookahead sprawdza, czy znaki lub wzorzec "x" nie występują bezpośrednio przed dopasowaniem danego wzorca.

Na przykład, (?!tr)a dopasuje "a", ale tylko jeśli przed nim nie występuje "tr". Dopasuje "ar", ale nie dopasuje "tra".

## Zadanie 1 - Walidacja adresu email

Stwórz wyrażenie regularne, które będzie dopasowywać poprawne adresy email. Adresy email mogą zawierać litery, cyfry, znaki "-" i "\_", a także znaki ".". Muszą mieć znak "@" oddzielający nazwę użytkownika od domeny.

## Zdanie 2 – Wyszukiwanie numerów telefonów

Stwórz wyrażenie regularne, które znajdzie numery telefonów w tekście. Numery mogą mieć różne formaty, np. (123) 456-7890 lub 123-456-7890.

## Zadanie 3 - Wyszukiwanie kodów pocztowych:

Stwórz wyrażenie regularne, które znajdzie kody pocztowe w tekście. Kody pocztowe mogą mieć różne formaty, np. 12-345 lub 12345.

## Zadanie 4 – Wyszukiwanie adresów: URL

Stwórz wyrażenie regularne, które znajdzie adresy URL w tekście. Adresy URL zaczynają się od "http://" lub "https://".

## Zadanie 5 - Wyszukiwanie dat w formacie MM/DD/YYYY

Stwórz wyrażenie regularne, które znajdzie daty w formacie MM/DD/YYYY w tekście.

## Zadanie 6 - Wyszukiwanie numerów zamówień

Stwórz wyrażenie regularne, które znajdzie numery zamówień w tekście. Numery zamówień mogą mieć różne formaty, np. "ORD12345" lub "INV-9876".