

https://www.tutorialspoint.com/perl/perl\_syntax.htm

*print ”Hello world\n”;*

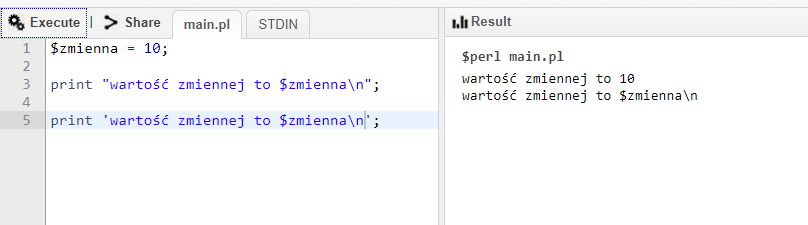
n jest po to żeby było przejście do kolejnej linijki

średnik jest na końcu każdego polecenia

PERL – practical extraction and report language

Larry Wall

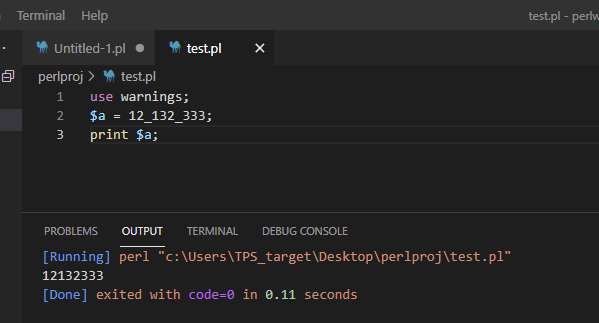
Jeśli w print użyjemy pojedynczego nawiasu zamiast podwójnego to zmienna będzie traktowana jako zwykły tekst:



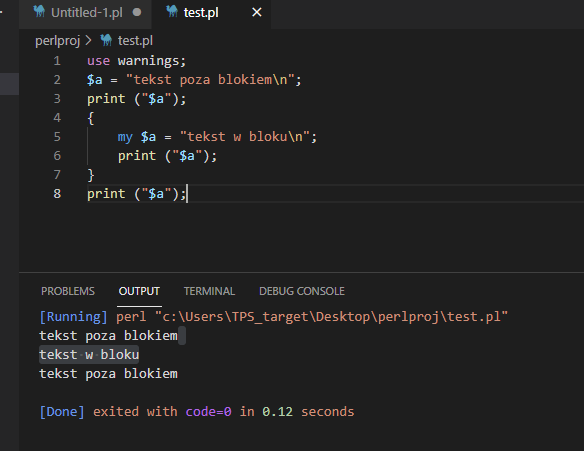
Używanie zmiennej w tekście (tak jak w pierwszym przykładzie) nazywa się interpolacją.

Interpolację można robić dla zmiennych, macierzy (arrays) ale nie dla klucz-warość(hashes).

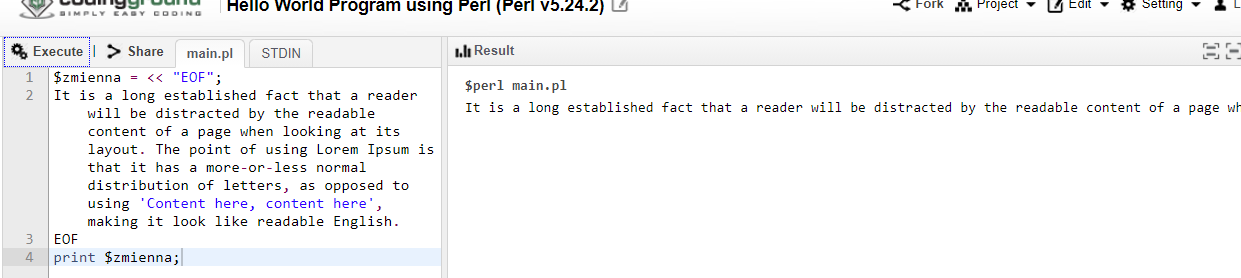
W celu zwiększenia czytelności zapisu duzych liczb możemy wstawiać znak \_ pomiędzy miejscami dziesiętnymi:



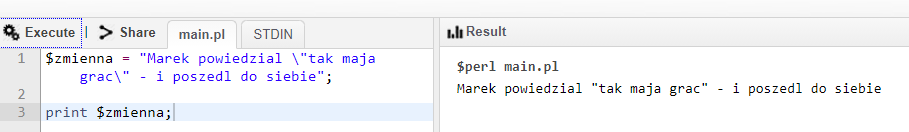
Zmienne mają swój zasięg – obejmują blok w którym się znajdują:



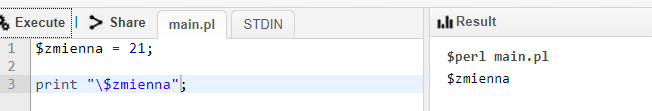
EOF – do przechowywania długich stringów



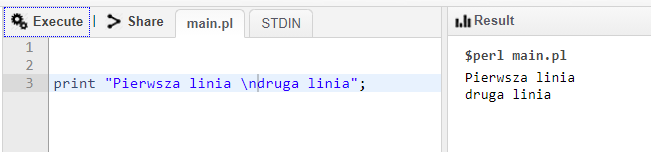
Żeby uwzględnić cudzysłów w stringu należy go poprzedzić znakiem \



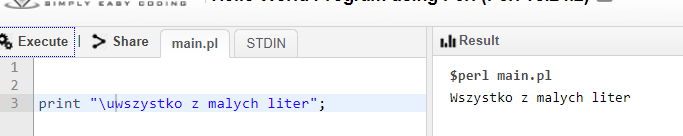
Jeśli użyjemy znaku \ przed $ to print potraktuje to jako tekst a nie jako zmienną:



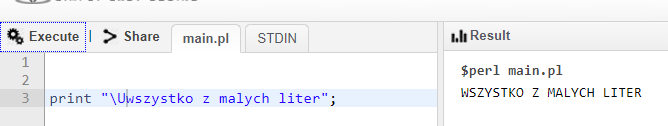
przejścia do nowej linii:

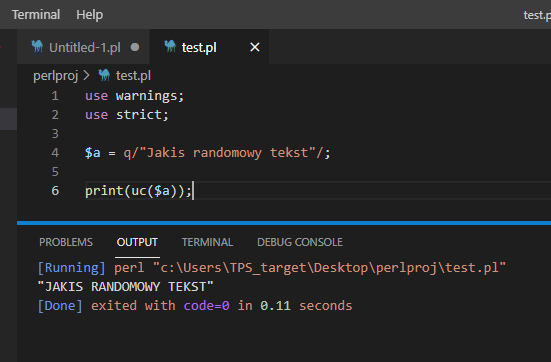


Wielka litera:

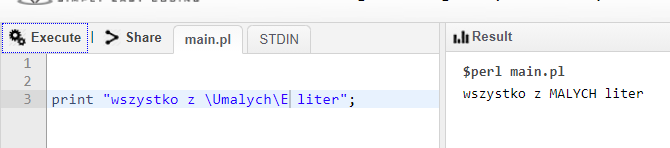


Wielka litera dla całego tekstu:

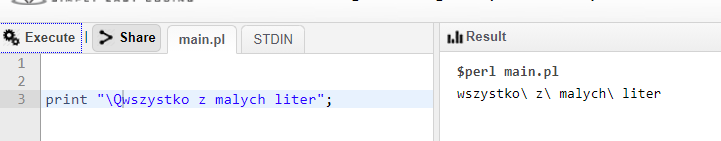




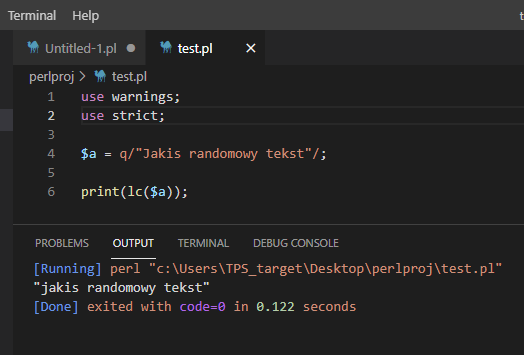
Wielka litera dla fragmentu tekstu – od \U do \E:



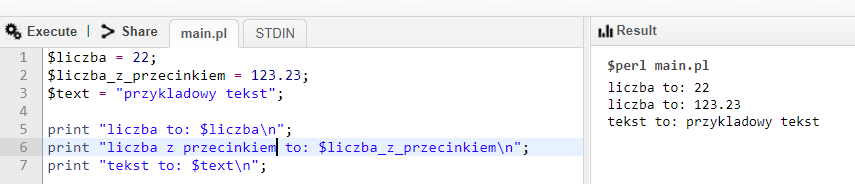
Znak \ przed każdym znakiem niealfanumerycznym (czyli np. spacją):

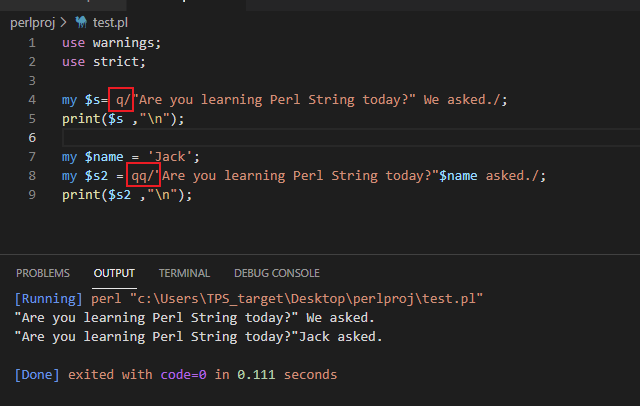


Mała litera dla całego tekstu:

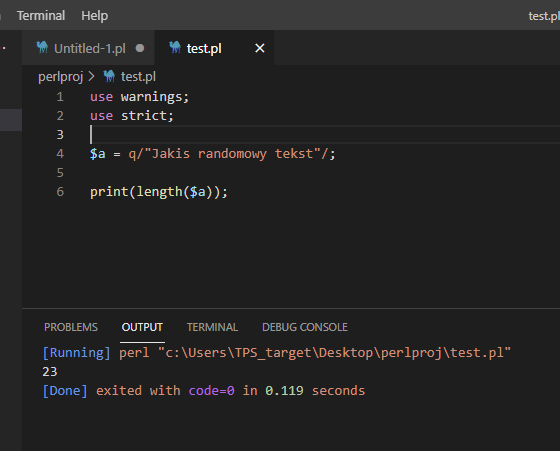


Typy danych – skalar. Może być liczbą albo tekstem:

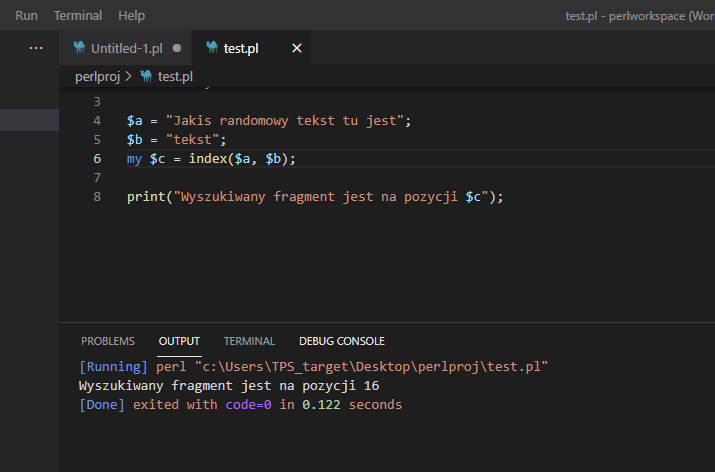


Tekst w zmiennej można zapisywać w inny sposób. W zależności czy będziemy robili interpolację czy nie możemy zapisać: 

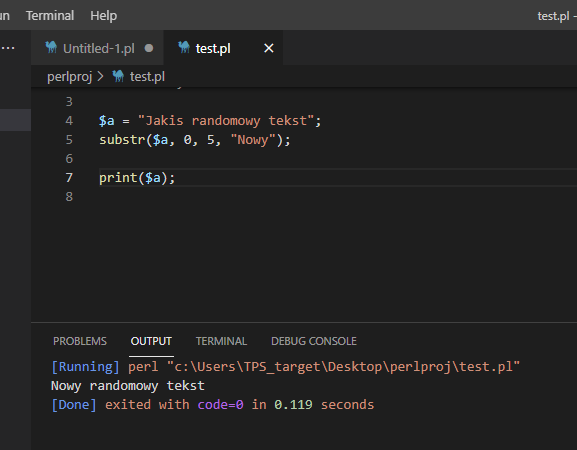
Sprawdzanie długości tekstu:



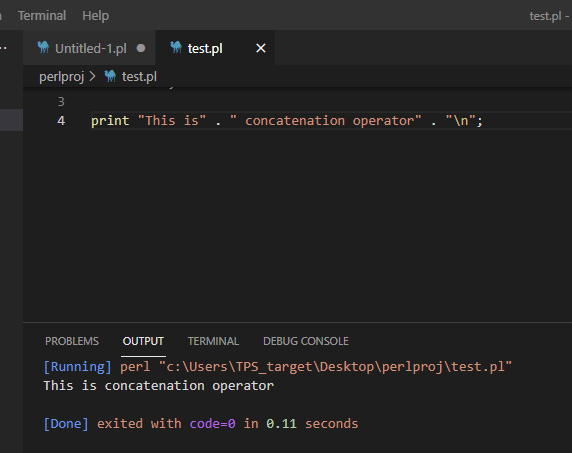
Wyszukiwanie tekstu w tekście:



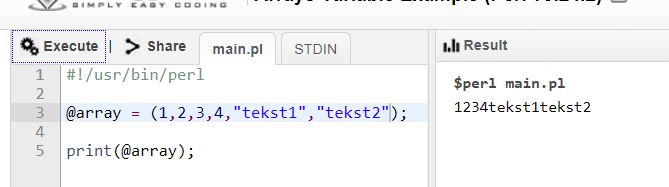
Zamiana franmentu tekstu:



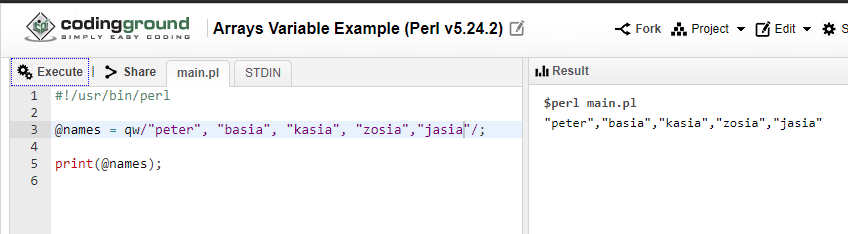
Konkatenacja:



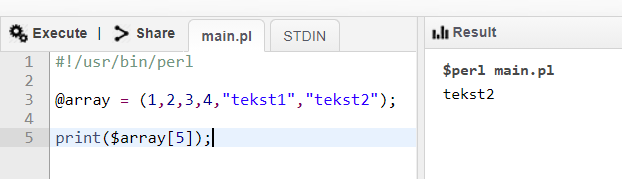
Typ danych – array czyli macierz (chyba):



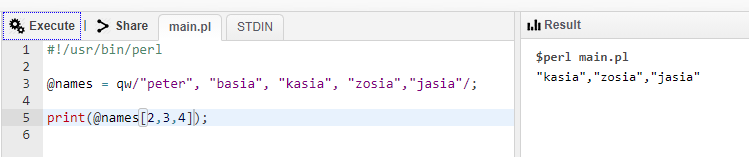
Inny sposób na deklarację macierzy:

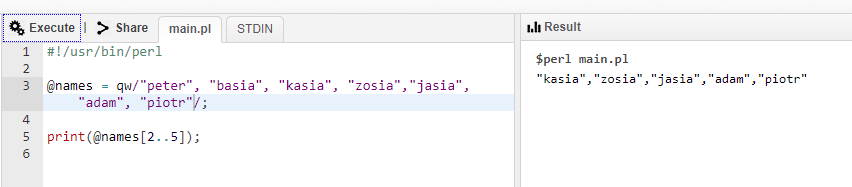


W taki sposób odwołujemy się do poszczególnych elementów macierzy:

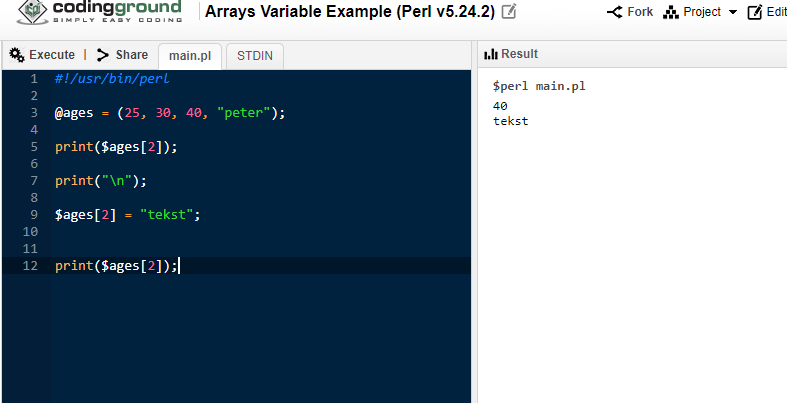


Taka lista która ma w sobie zarówno stringi jak i numery nazywa się complex list

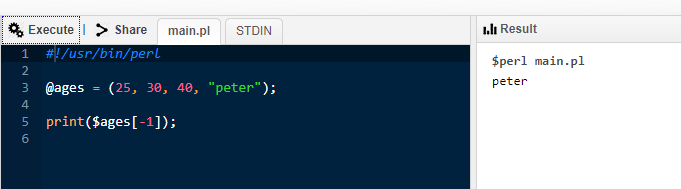




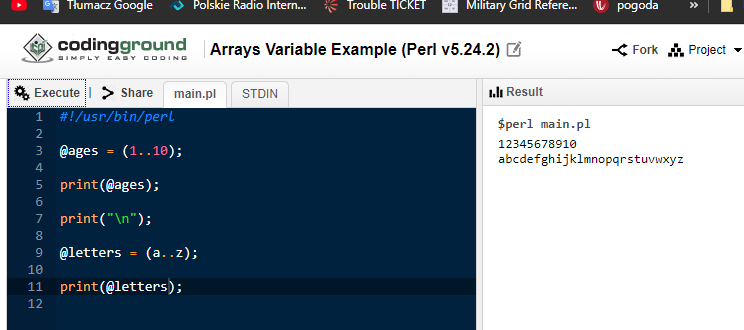
Wstawianie wartości do zmiennych w macierzy:



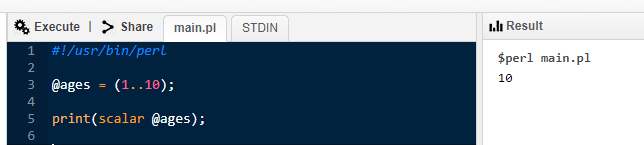
Można odwoływać się do elementów macierzy od tyłu:

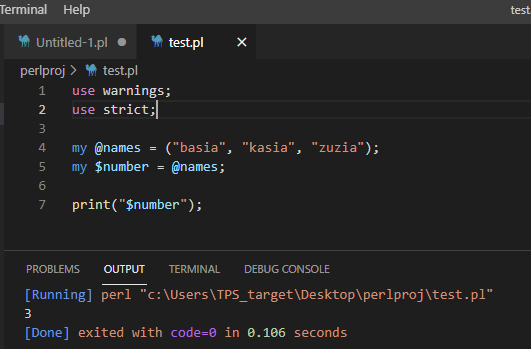


Sekwencyjne tworzenie macierzy:

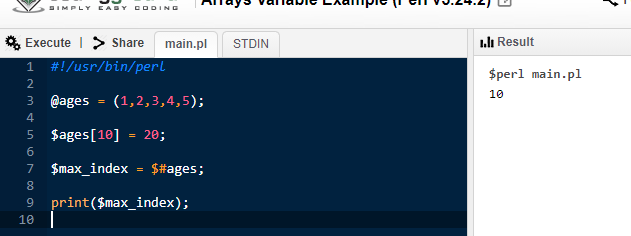


Sprawdzanie rozmiaru macierzy:

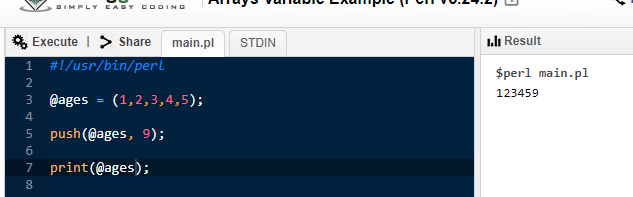




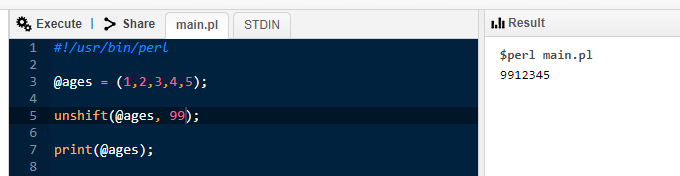
Macierz nie musi być „ciągła”. Możemy sprawdzić maksymalny indeks macierzy:



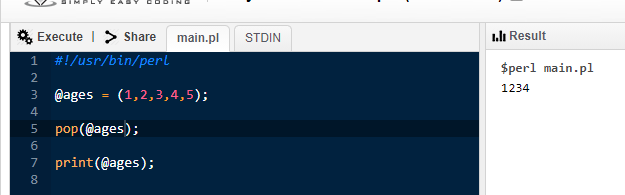
Dodawanie elementów na końcu macierzy:



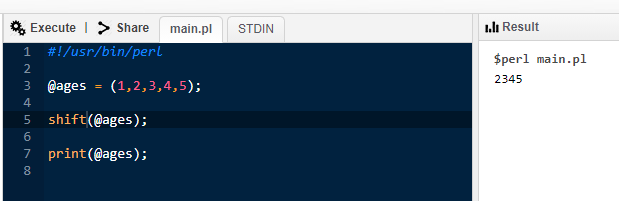
Dodawanie elementu na początku macierzy:



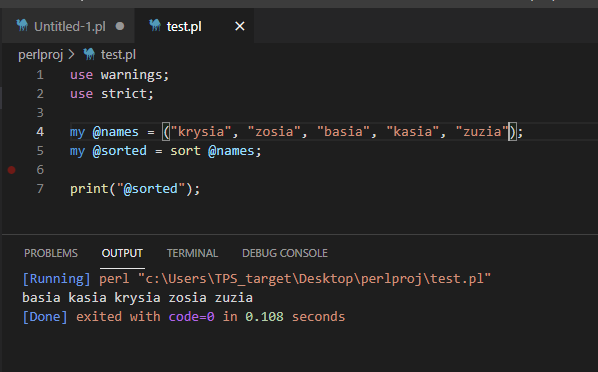
Usuwanie ostatniego elementu listy:



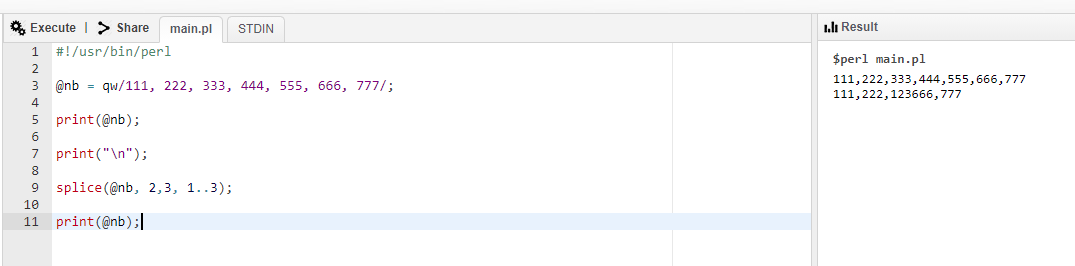
Usuwanie pierwszego elementu listy:



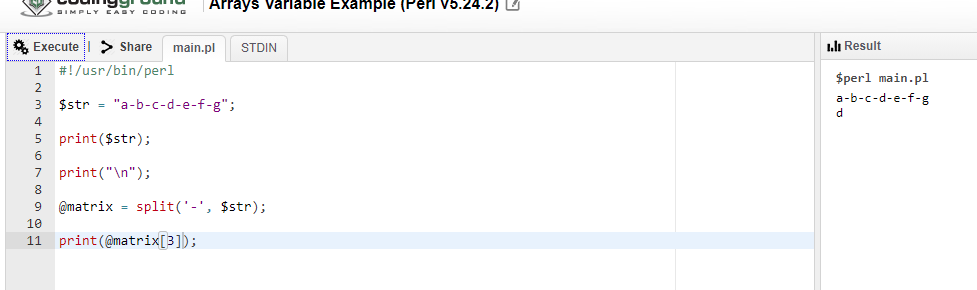
Sortowanie elementów w macierzy:



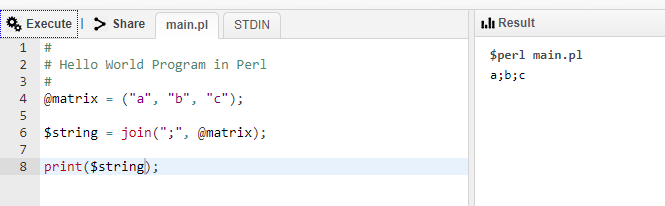
Zastępowanie wybranych elementów macierzy:



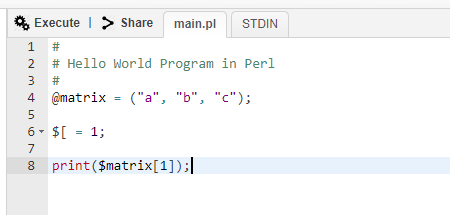
Zmiana stringa w macierz:



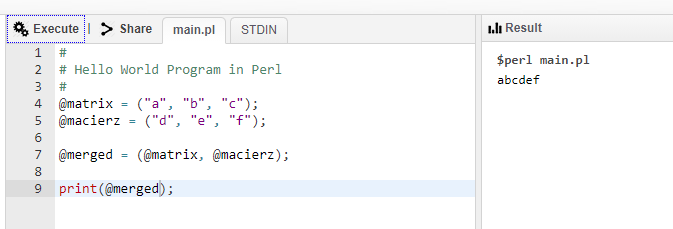
Zmiana macierzy w string:



Macierze są numerowane od 0 ale można zmienić to na inną wartość:

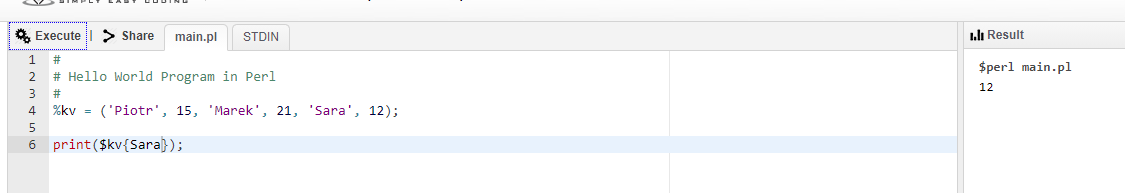


Łączenie macierzy:

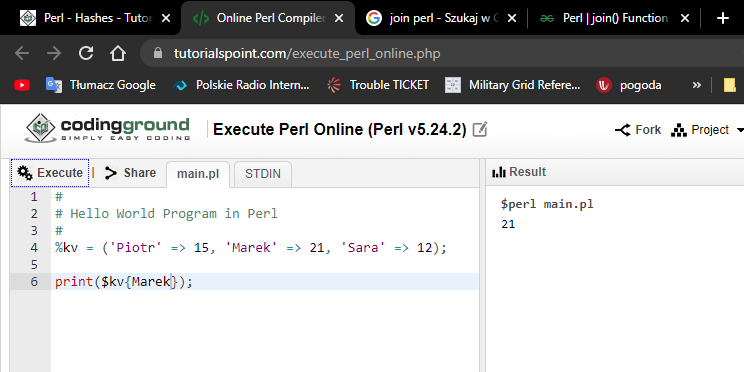


Typ danych klucz-wartość:

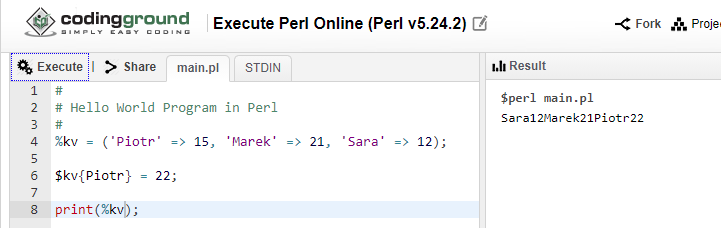
Deklaracja i odwołanie do wartości:



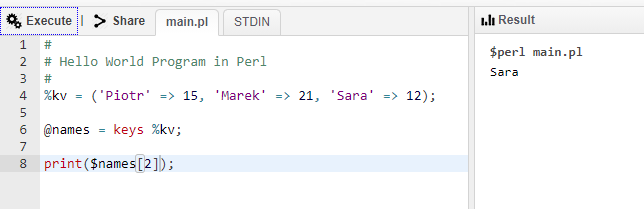
Można też deklarować w taki sposób żeby nie było wątpliwości co jest kluczem a co wartością:

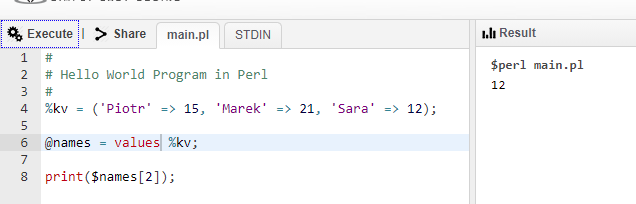


Zmiana wartości:

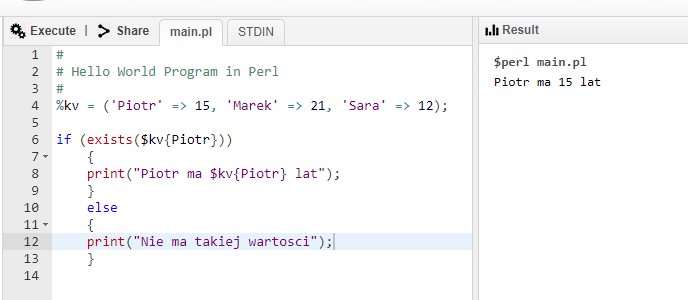


Odwołanie do kv jak do macierzy wywyołując klucz lub wartość:

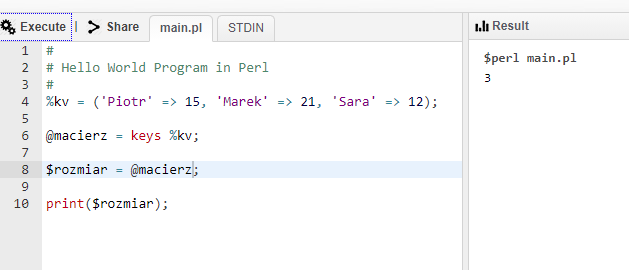




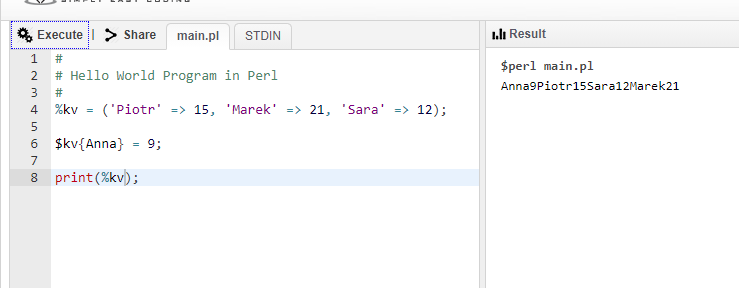
Sprawdzanie czy podany klucz istnieje i funkcja if:



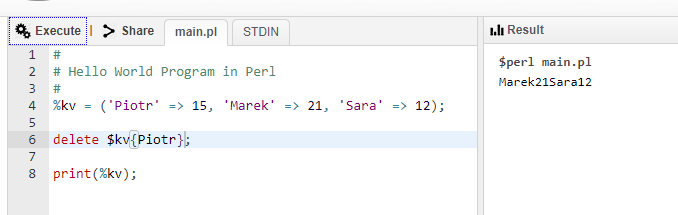
Sprawdzane rozmiaru kv:



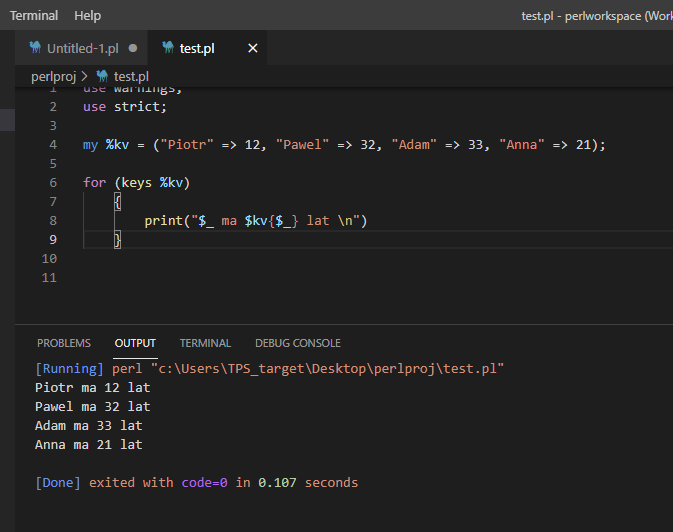
Dodawanie kolejnej wartości do kv:



Usuwanie wartości z kv:

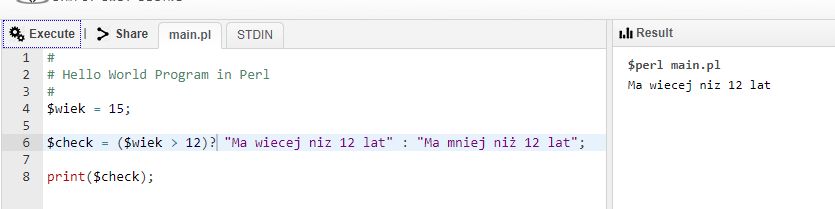


Przelatywanie po wartościach w kv:

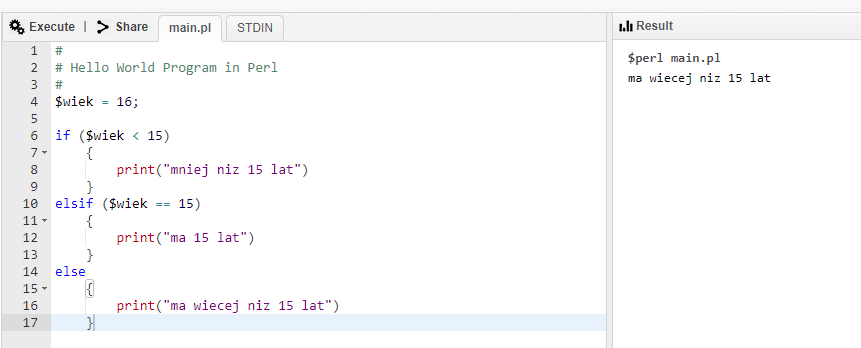


If than else:

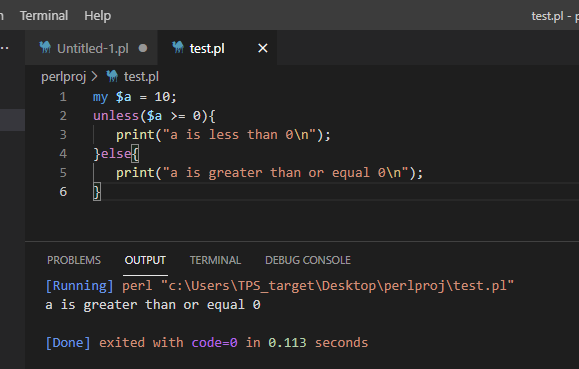
Najprostrza forma:



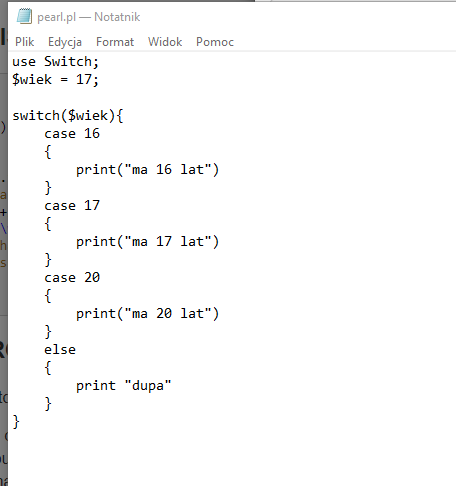
Zwykła forma:

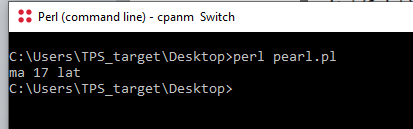


Jest jeszcze unless, który działa podobnie:

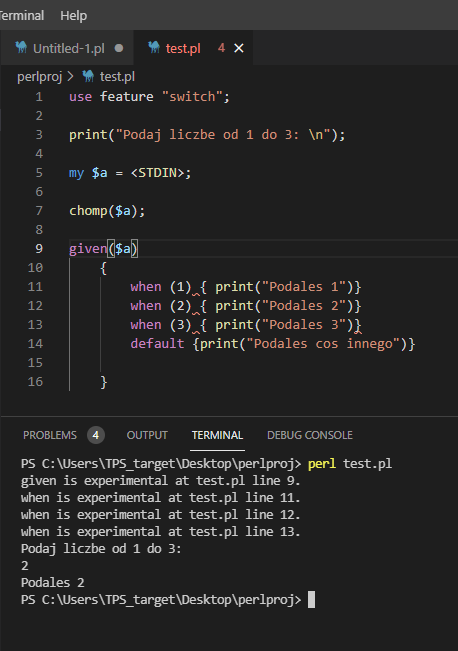


Switch:

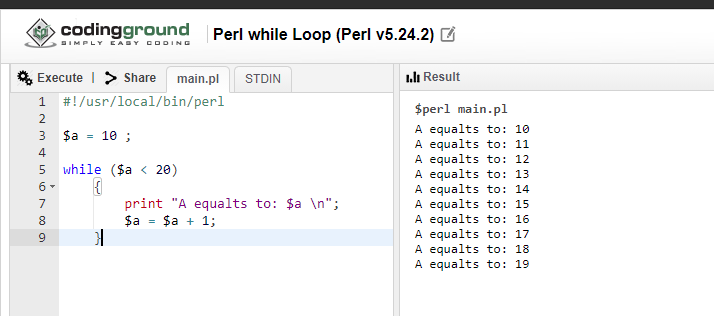




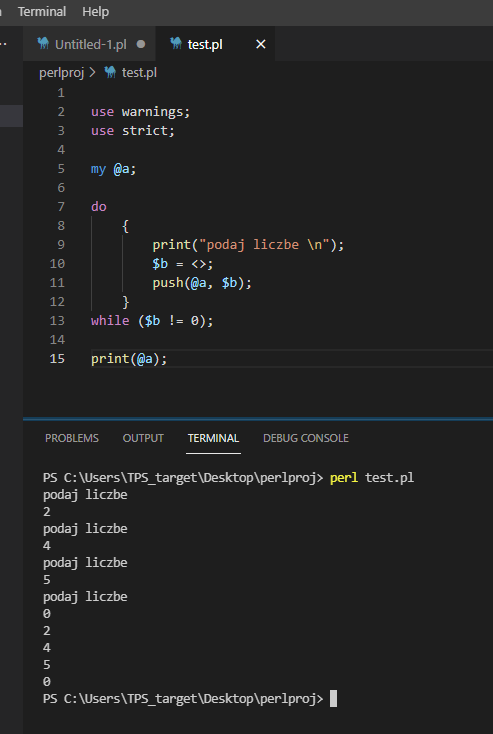
Alternatywą do switch jest given:



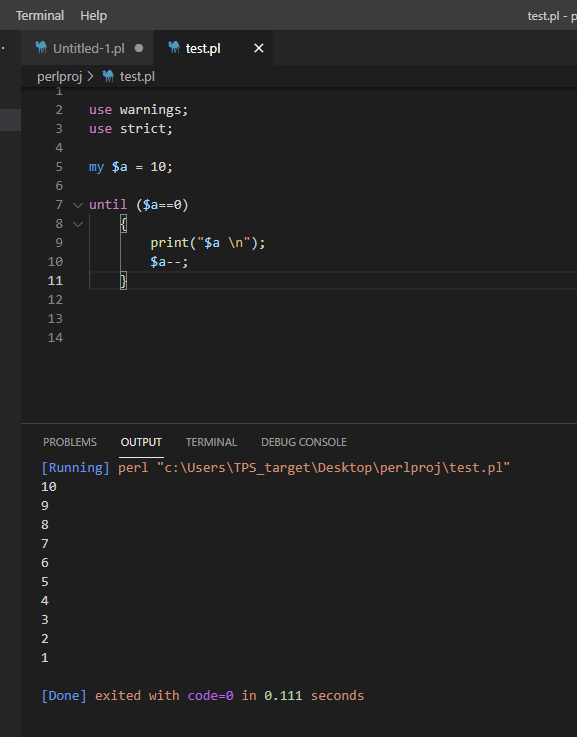
Pętla while:



Do while:



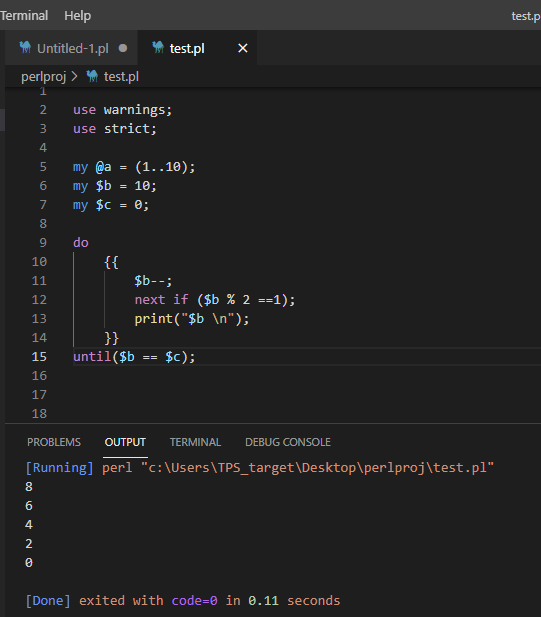
Until:



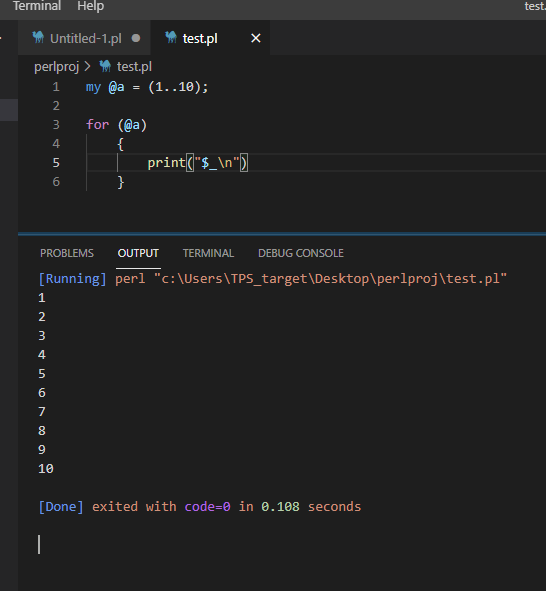
Do until:



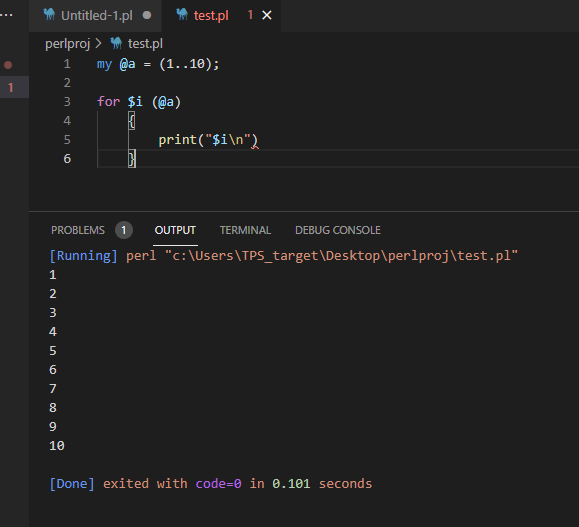
Next czyli omijanie linijek kodu w przypadku dgy spełniony jest określony warunek:

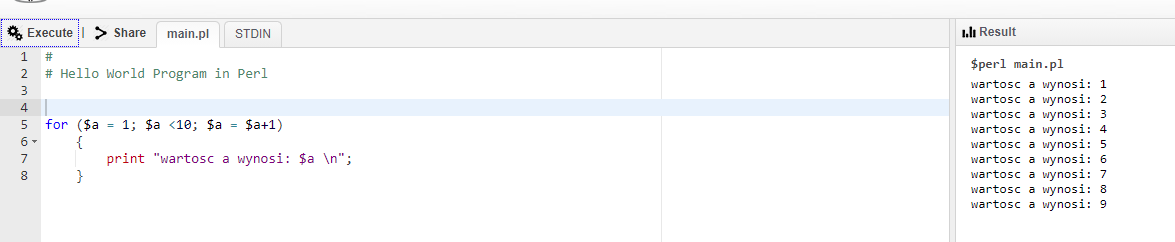


Pętla for:

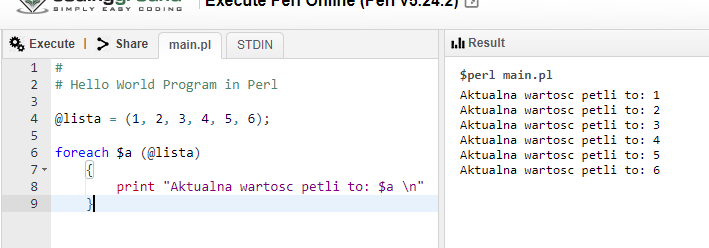


Wykorzystujemy tutaj przypisanie do specjalnej zmiennej $\_ ale możemy też sobie stworzyć sztuczny iterator:

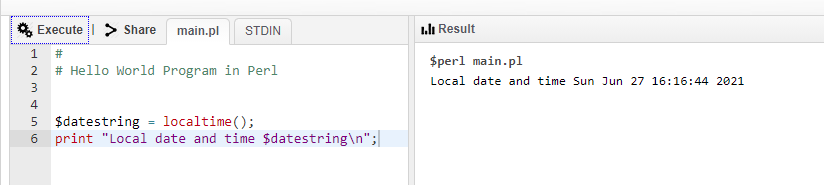


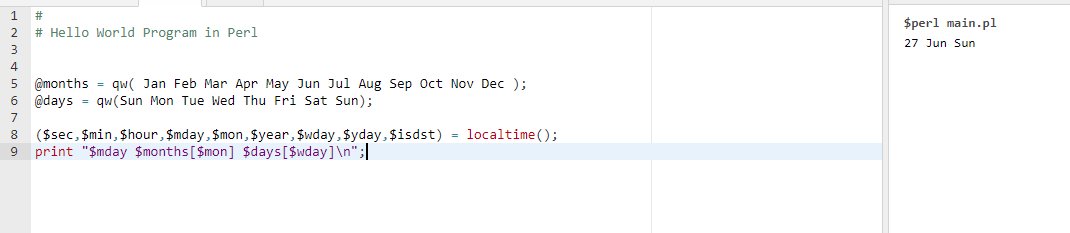


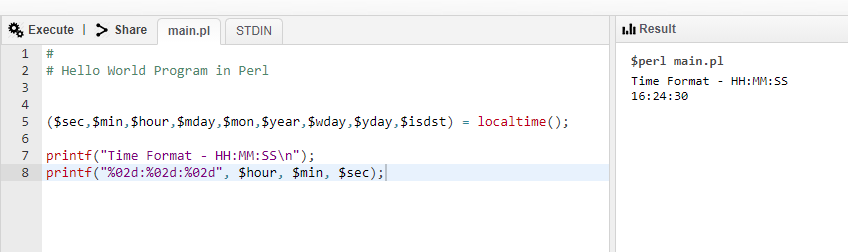
Pętla foreach:



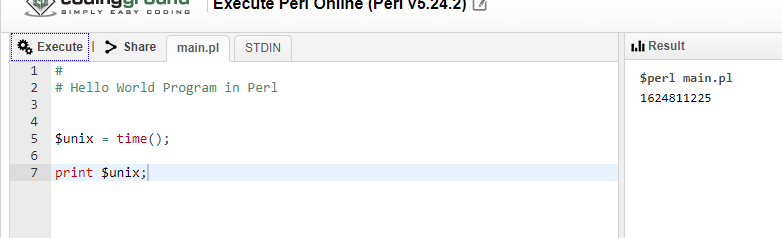
Data:



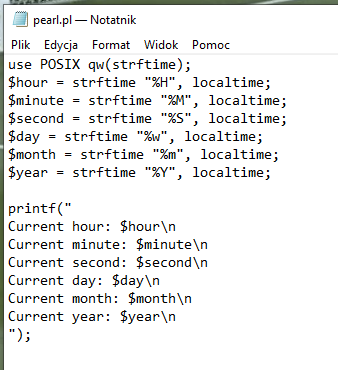


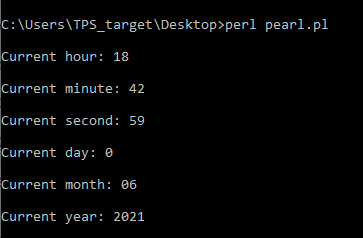


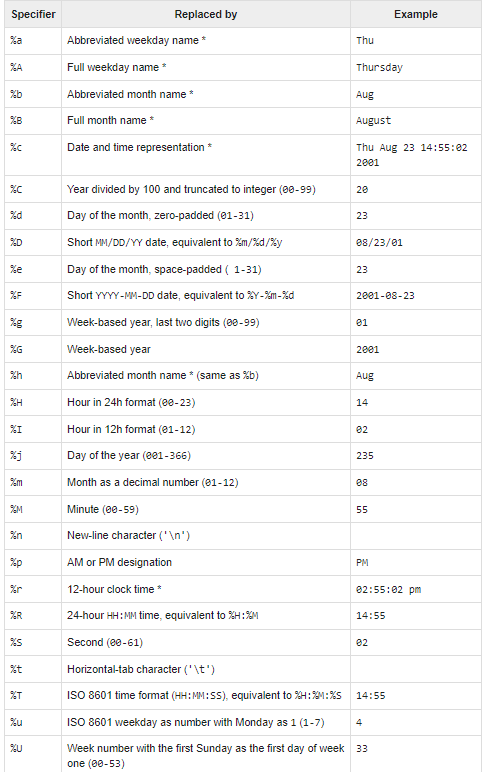
Czas unixowy:

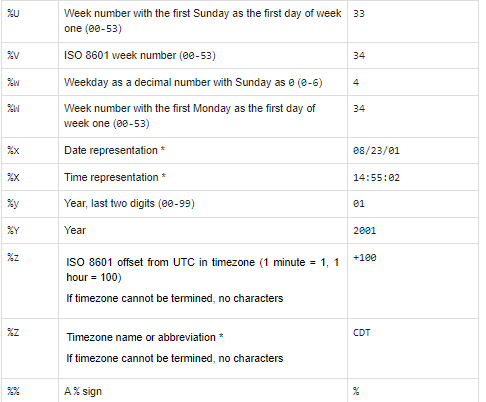


Maski daty:



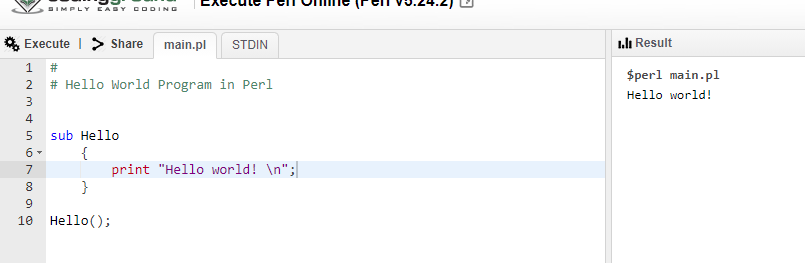




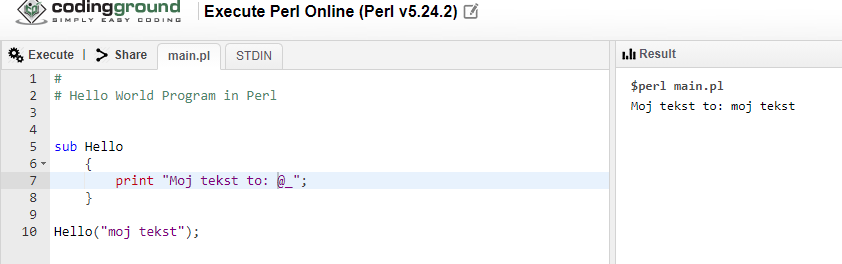


Subroutines

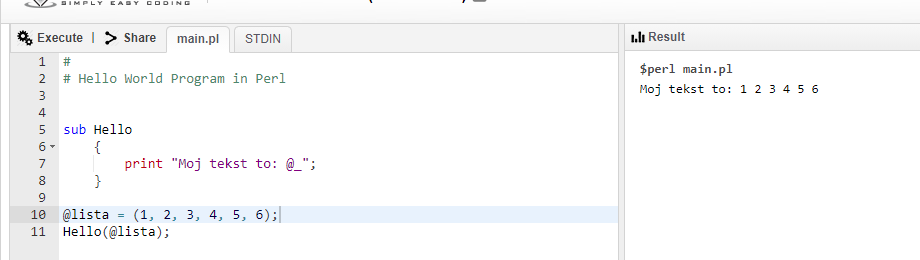
Coś jak funkcja:



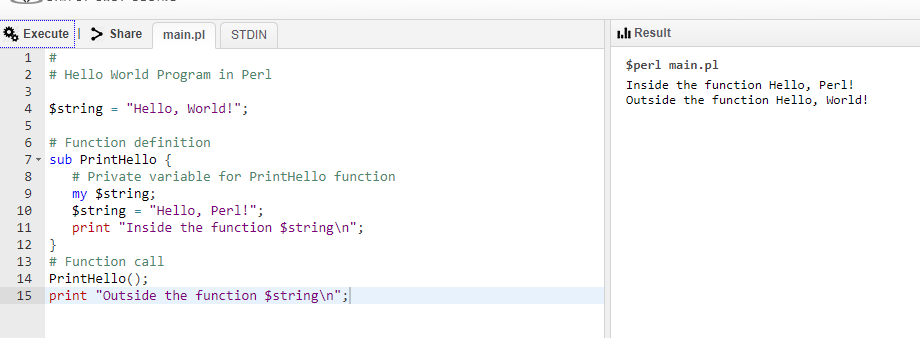
Funkcja z parametrem:



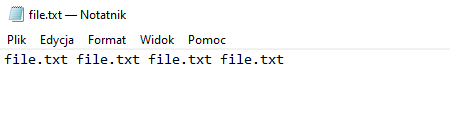
Jako argument można podawać listę:

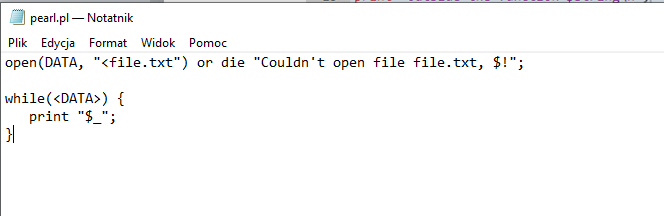


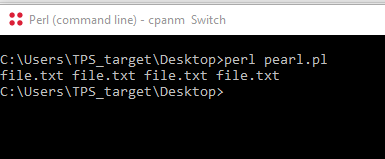
Zmienne prywatne i globalne:



Odczyt z pliku txt umiezczonego w tej samej lokalizacji co odpalany:



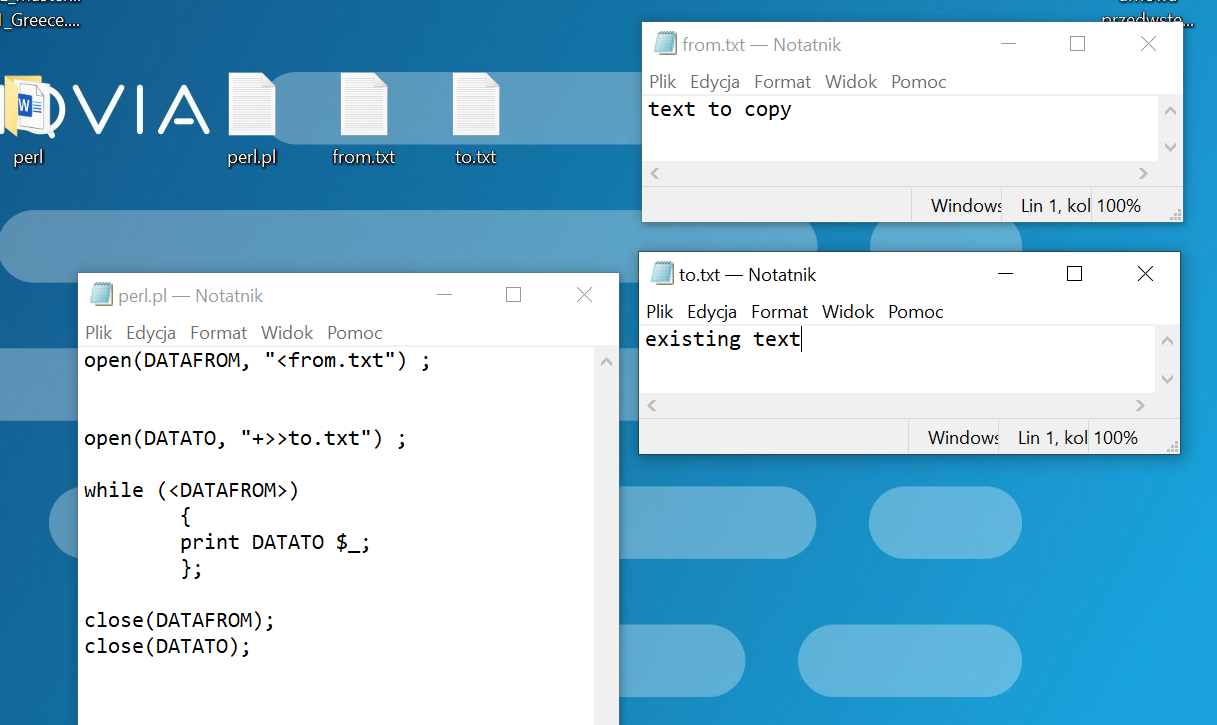




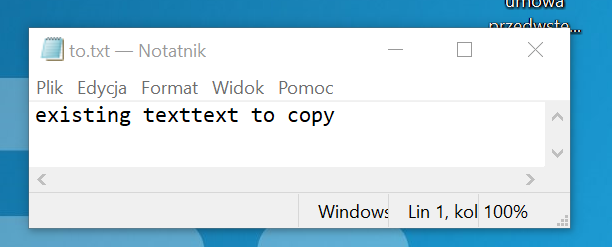
open(DATA, "<file.txt")

Zaznaczony fragment odpowiada za to, że odpalamy plik w trybie read only.

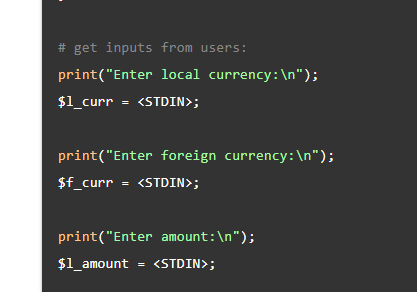
Kopiowanie z pliku do pliku:



Zawartość pliku po odpaleniu skryptu:



Wprowadzanie wartości z terminala do zmiennych:

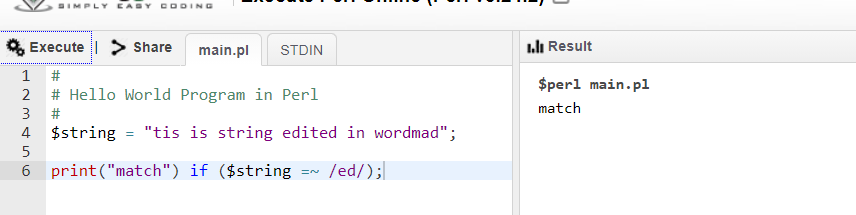


Sprawdzanie czy w podanym tekście są cyfry lub litery:

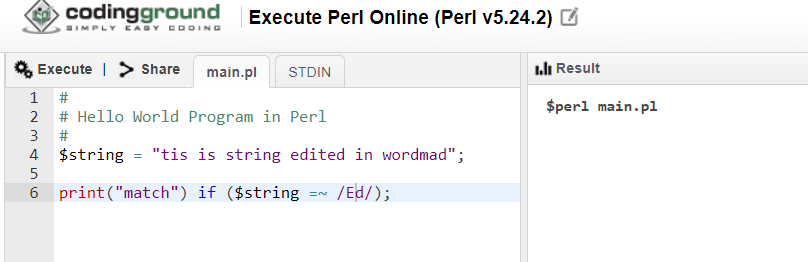


Regular expressions:

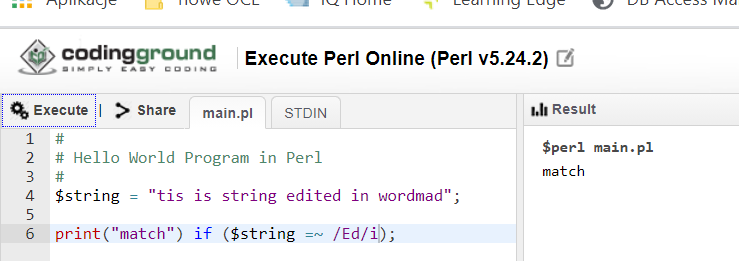
Sprawdzamy czy w podanym tekście jest string ‘ed’:



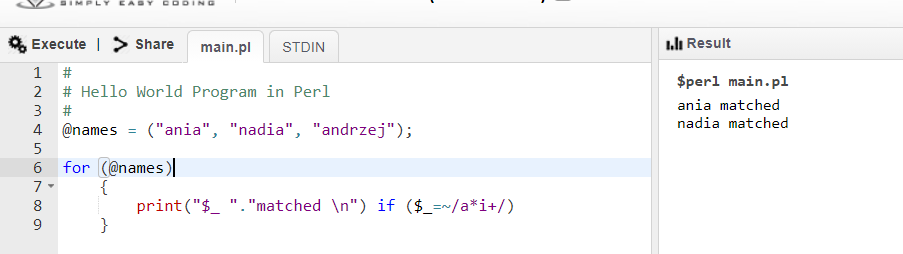
Wielkość znaków ma znaczenie. W takim przypadku „Ed” nie zostanie wyszukane:

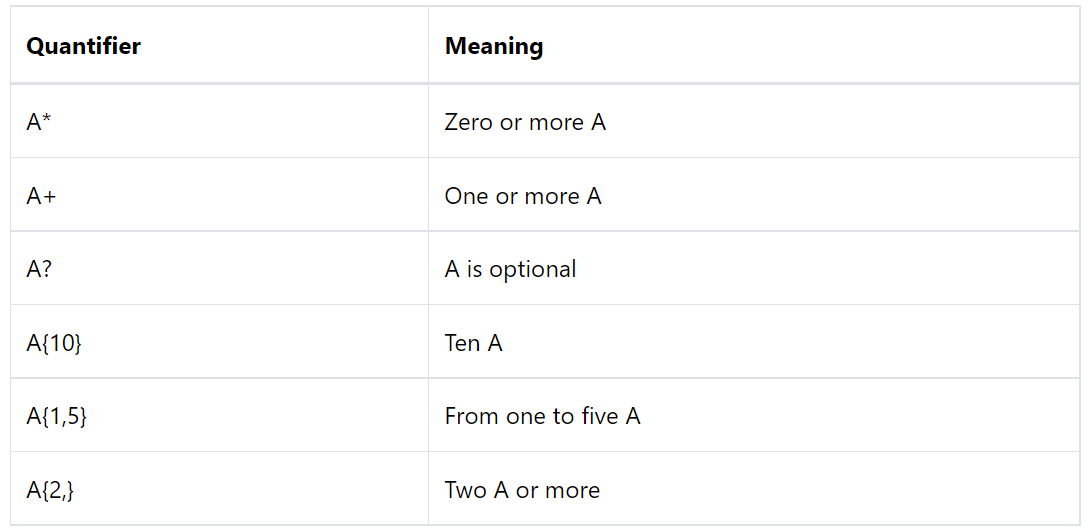


Ale jeśli zrobimy w ten sposób to wielkość znaków przestanie mieć znaczenie:

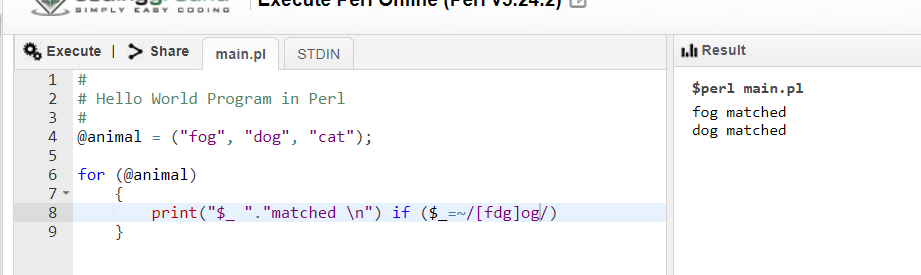


Teraz wyszukujemy w ciągu wszystkich jego elementów w którym występuje zero lub więcej liter „a”, po których jest co najmniej jedna litera „i”:





W nawiasach klamrowych umieszczamy zbiór możliwych liter:

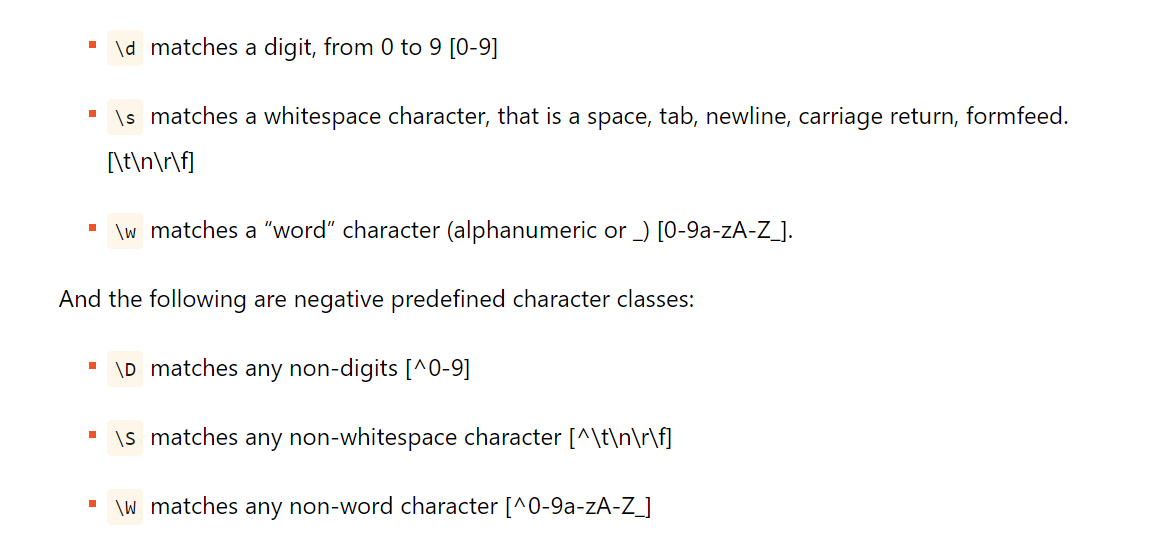


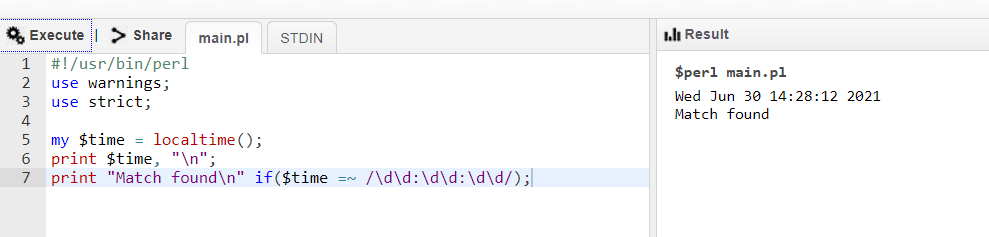
Czyli w tym przypadku wyszukane mogą być wyrazy: „fog”, „dog”, „gog”.

/elem[012345]/; # matches 'elem0', 'elem1'... or 'elem5'

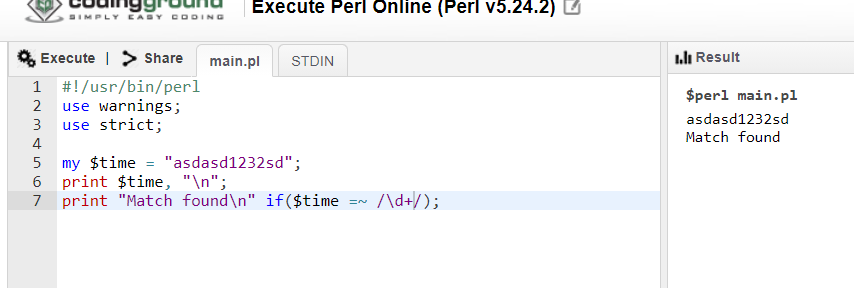
/[elem[0-1000]]/;

/[a-z]1/; matches a1,b1,... to z1

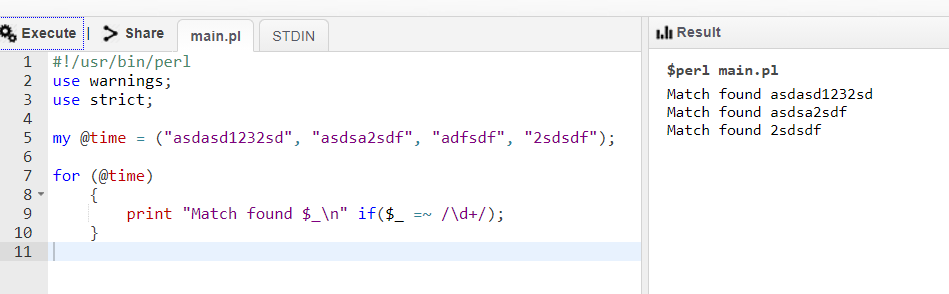




Sprawdzamy czy w stringu są cyfry:



To samo tylko z macierzą:



https://www.perltutorial.org/perl-regular-expression-character-classes/