

Narzędzia informatyczne w językoznawstwie

Perl - Tablice asocjacyjne oraz funkcje tablicowe

Marcin Junczys-Dowmunt

`junczys@amu.edu.pl`

Zakład Logiki Stosowanej

<http://www.logic.amu.edu.pl>

19. grudnia 2007

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów
 - ▶ sposoby dodawania i usuwanie elementów

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów
 - ▶ sposoby dodawania i usuwanie elementów
 - ▶ sposoby przeglądania haszów

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów
 - ▶ sposoby dodawania i usuwanie elementów
 - ▶ sposoby przeglądania haszów
- ▶ Omówimy powiązania między haszami i tablicami

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów
 - ▶ sposoby dodawania i usuwanie elementów
 - ▶ sposoby przeglądania haszów
- ▶ Omówimy powiązania między haszami i tablicami
 - ▶ w tym sortowanie elementów tablic i haszów za pomocą `sort`

- ▶ Skupimy się na jednym z najpotężniejszych narzędzi w Perlu – na tzw. *haszach*:
 - ▶ omówimy istotę haszów
 - ▶ sposoby inicjalizacji haszów
 - ▶ sposoby dodawania i usuwanie elementów
 - ▶ sposoby przeglądania haszów
- ▶ Omówimy powiązania między haszami i tablicami
 - ▶ w tym sortowanie elementów tablic i haszów za pomocą `sort`
 - ▶ tworzenie np. haszów z tablic za pomocą `map`

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości
- ▶ Kluczem hasza może być dowolna wartość skalarna, czyli liczba, łańcuch znakowy (lub referencja)

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości
- ▶ Kluczem hasza może być dowolna wartość skalarna, czyli liczba, łańcuch znakowy (lub referencja)
- ▶ Przedrostkiem dla hasza jest znak %, który ma przypominać parę elementów skojarzonych

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości
- ▶ Kluczem hasza może być dowolna wartość skalarna, czyli liczba, łańcuch znakowy (lub referencja)
- ▶ Przedrostkiem dla hasza jest znak %, który ma przypominać parę elementów skojarzonych
- ▶ Zamiast nawiasów [] korzystamy z {} przy odwoływaniu się do wartości haszów

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości
- ▶ Kluczem hasza może być dowolna wartość skalarna, czyli liczba, łańcuch znakowy (lub referencja)
- ▶ Przedrostkiem dla hasza jest znak %, który ma przypominać parę elementów skojarzonych
- ▶ Zamiast nawiasów [] korzystamy z {} przy odwoływaniu się do wartości haszów
- ▶ Hasze są jedną z najpotężniejszych i najczęściej używanych cech Perla

Co to jest hasz?

- ▶ Hasz jest strukturą podobną do tablicy, ale zamiast indeksów liczbowych hasz używa kluczy
- ▶ Bardziej skomplikowana nazwa dla haszy to *tablice asocjacyjne*, ponieważ kojarzą ze sobą klucze i wartości
- ▶ Kluczem hasza może być dowolna wartość skalarna, czyli liczba, łańcuch znakowy (lub referencja)
- ▶ Przedrostkiem dla hasza jest znak %, który ma przypominać parę elementów skojarzonych
- ▶ Zamiast nawiasów [] korzystamy z {} przy odwoływaniu się do wartości haszów
- ▶ Hasze są jedną z najpotężniejszych i najczęściej używanych cech Perla
- ▶ Ponoć by programować w Perlu, trzeba myśleć w haszach

Inicjalizacja haszów

```
1 my %dzwieki = (  
    lew => "grrr",  
    pies => "hau",  
    kot => "miau",  
5    "tygrys bengalski" => "roar"  
);  
  
print "Lew robi ". $dzwieki{"lew"} ."\n";  
print "Pies robi ". $dzwieki{pies} ."\n";  
10 print "Kot robi $dzwieki{kot}\n";  
print "Tygrys robi $dzwieki{'tygrys bengalski'}\n";
```

Inicjalizacja haszów

```
1 my %dzwieki = (  
    lew => "grrr",  
    pies => "hau",  
    kot => "miau",  
5    "tygrys bengalski" => "roar"  
);  
  
print "Lew robi ". $dzwieki{"lew"} ."\n";  
print "Pies robi ". $dzwieki{pies} ."\n";  
10 print "Kot robi $dzwieki{kot}\n";  
print "Tygrys robi $dzwieki{'tygrys bengalski'}\n";
```

- ▶ Kanoniczny sposób inicjalizacji haszów

Inicjalizacja haszów

```
1 my %dzwieki = (  
    lew => "grrr",  
    pies => "hau",  
    kot => "miau",  
5    "tygrys bengalski" => "roar"  
);  
  
print "Lew robi ". $dzwieki{"lew"} ."\n";  
print "Pies robi ". $dzwieki{pies} ."\n";  
10 print "Kot robi $dzwieki{kot}\n";  
print "Tygrys robi $dzwieki{'tygrys bengalski'}\n";
```

- ▶ Kanoniczny sposób inicjalizacji haszów
- ▶ Kojarzemy ze sobą nazwy zwierząt oraz wydawane dźwięki

Inicjalizacja haszów

```
1 my %dzwieki = (  
    lew => "grrr",  
    pies => "hau",  
    kot => "miau",  
5    "tygrys bengalski" => "roar"  
);  
  
print "Lew robi ". $dzwieki{"lew"} ."\n";  
print "Pies robi ". $dzwieki{pies} ."\n";  
10 print "Kot robi $dzwieki{kot}\n";  
print "Tygrys robi $dzwieki{'tygrys bengalski'}\n";
```

- ▶ Kanoniczny sposób inicjalizacji haszów
- ▶ Kojarzemy ze sobą nazwy zwierząt oraz wydawane dźwięki
- ▶ Gdy klucz składa się z samych znaków alfanumerycznych, możemy opuścić cudzysłów

Dodawanie elementów do haszów

```
1 my %oczy;
   $oczy{kot} = "2";

   @oczy{mrowka, waszka} = (4, "milion");
5
   my @stwory = qw(pantofelek, czlowiek, pajak, mucha);
   @oczy{@stwory} = (0, 2, 8, "duzo za duzo");
```

Dodawanie elementów do haszów

```
1 my %oczy;  
  $oczy{kot} = "2";  
  
  @oczy{mrowka, waszka} = (4, "milion");  
5  
my @stwory = qw(pantofelek, czlowiek, pajak, mucha);  
@oczy{@stwory} = (0, 2, 8, "duzo za duzo");
```

- ▶ Możemy dodawać dowolną liczbą elementów do hasza

Dodawanie elementów do haszów

```
1 my %oczy;  
  $oczy{kot} = "2";  
  
  @oczy{mrowka, waszka} = (4, "milion");  
5  
my @stwory = qw(pantofelek, czlowiek, pajak, mucha);  
@oczy{@stwory} = (0, 2, 8, "duzo za duzo");
```

- ▶ Możemy dodawać dowolną liczbą elementów do hasza
- ▶ Ponieważ nie ma określonej kolejności elementów w haszu, nie potrzebujemy funkcji typu unshift, push

Dodawanie elementów do haszów

```
1 my %oczy;  
  $oczy{kot} = "2";  
  
  @oczy{mrowka, waszka} = (4, "milion");  
5  
my @stwory = qw(pantofelek, czlowiek, pajak, mucha);  
@oczy{@stwory} = (0, 2, 8, "duzo za duzo");
```

- ▶ Możemy dodawać dowolną liczbą elementów do hasza
- ▶ Ponieważ nie ma określonej kolejności elementów w haszu, nie potrzebujemy funkcji typu unshift, push
- ▶ Podobnie jak dla tablic istnieją wycinki haszów

Dodawanie elementów do haszów

```
1 my %oczy;  
   $oczy{kot} = "2";  
  
   @oczy{mrowka, waszka} = (4, "milion");  
5  
my @stwory = qw(pantofelek, czlowiek, pajak, mucha);  
@oczy{@stwory} = (0, 2, 8, "duzo za duzo");
```

- ▶ Możemy dodawać dowolną liczbą elementów do hasza
- ▶ Ponieważ nie ma określonej kolejności elementów w haszu, nie potrzebujemy funkcji typu `unshift`, `push`
- ▶ Podobnie jak dla tablic istnieją wycinki haszów
- ▶ Każdy wycinek z hasza jest tablicą(!), stąd przedrostek `@`

Usuwanie elementów z hasza

```
1 my %hasz = (klucz1 => 1, klucz2 => 2, klucz3 => 3);  
  # my %hasz = ("klucz", 1, "klucz2", 2, "klucz3", 3);  
  
my $skalar1 = delete $hasz{klucz1};  
5 # $skalar1 równy 1  
  
my $skalar2 = delete @hash{qw(klucz1 klucz2)};  
  # skalar2 równy 2  
  
10 @tablica = delete @hash{qw(klucz1 klucz2 klucz3)};  
  # @tablica równa (undef, undef, 3)
```

Usuwanie elementów z hasza

```
1 my %hasz = (klucz1 => 1, klucz2 => 2, klucz3 => 3);  
  # my %hasz = ("klucz", 1, "klucz2", 2, "klucz3", 3);  
  
my $skalar1 = delete $hasz{klucz1};  
5 # $skalar1 równy 1  
  
my $skalar2 = delete @hash{qw(klucz1 klucz2)};  
  # skalar2 równy 2  
  
10 @tablica = delete @hash{qw(klucz1 klucz2 klucz3)};  
  # @tablica równa (undef, undef, 3)
```

- ▶ Funkcja wbudowana delete usuwa podane elementy z hasza

Usuwanie elementów z hasza

```
1 my %hasz = (klucz1 => 1, klucz2 => 2, klucz3 => 3);  
  # my %hasz = ("klucz", 1, "klucz2", 2, "klucz3", 3);  
  
my $skalar1 = delete $hasz{klucz1};  
5 # $skalar1 równy 1  
  
my $skalar2 = delete @hash{qw(klucz1 klucz2)};  
  # skalar2 równy 2  
  
10 @tablica = delete @hash{qw(klucz1 klucz2 klucz3)};  
  # @tablica równa (undef, undef, 3)
```

- ▶ Funkcja wbudowana delete usuwa podane elementy z hasza
- ▶ W kontekście skalarnym zwraca ostatni usunięty element lub undef jeśli element nie istnieje

Usuwanie elementów z hasza

```
1 my %hasz = (klucz1 => 1, klucz2 => 2, klucz3 => 3);  
  # my %hasz = ("klucz", 1, "klucz2", 2, "klucz3", 3);  
  
my $skalar1 = delete $hasz{klucz1};  
5 # $skalar1 równy 1  
  
my $skalar2 = delete @hash{qw(klucz1 klucz2)};  
  # skalar2 równy 2  
  
10 @tablica = delete @hash{qw(klucz1 klucz2 klucz3)};  
  # @tablica równa (undef, undef, 3)
```

- ▶ Funkcja wbudowana delete usuwa podane elementy z hasza
- ▶ W kontekście skalarnym zwraca ostatni usunięty element lub undef jeśli element nie istnieje
- ▶ W kontekście listowym zwraca listę wszystkich usuniętych elementów, w tym undef, jeśli jakiś z elementów nie istnieje

Sprawdzanie czy element istnieje w haszu

```
1  my %mity = (  
    yeti => 0,  
    gwiazdor => "",  
    wilkolak => undef  
5  );  
  
chomp(my $test = <STDIN>);  
if(exists($mity{$test})) {  
    print "$test istnieje, wartość '$mity{$test}'\n";  
10 }  
else {  
    print "$test nie istnieje\n";  
}
```

Sprawdzanie czy element istnieje w haszu

```
1 my %mity = (  
    yeti => 0,  
    gwiazdor => "",  
    wilkolak => undef  
5 );  
  
chomp(my $test = <STDIN>);  
if(exists($mity{$test})) {  
    print "$test istnieje, wartość '$mity{$test}'\n";  
10 }  
else {  
    print "$test nie istnieje\n";  
}
```

- ▶ Funkcja `exists` sprawdza, czy dany element istnieje w haszu

Sprawdzanie czy element istnieje w haszu

```
1 my %mity = (  
    yeti => 0,  
    gwiazdor => "",  
    wilkolak => undef  
5 );  
  
chomp(my $test = <STDIN>);  
if(exists($mity{$test})) {  
    print "$test istnieje, wartość '$mity{$test}'\n";  
10 }  
else {  
    print "$test nie istnieje\n";  
}
```

- ▶ Funkcja `exists` sprawdza, czy dany element istnieje w haszu
- ▶ Istnienie takiej funkcji jest konieczne, ponieważ wartość skojarzona z danym kluczem może być logicznym fałszem

Przeglądanie haszów - według kluczy

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (keys %słownik) {  
    print "Wyraz $_ to $słownik{$_}\n";  
}
```

Przeglądanie haszów - według kluczy

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (keys %słownik) {  
    print "Wyraz $_ to $słownik{$_}\n";  
}
```

- ▶ Funkcja wbudowana keys zwraca listę wszystkich kluczy danego hasza

Przeglądanie haszów - według kluczy

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (keys %słownik) {  
    print "Wyraz $_ to $słownik{$_}\n";  
}
```

- ▶ Funkcja wbudowana `keys` zwraca listę wszystkich kluczy danego hasza
- ▶ Można tę listę zapisać do zmiennej tablicowej lub użyć w dowolnym kontekście listowym

Przeglądanie haszów - według kluczy

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (keys %słownik) {  
    print "Wyraz $_ to $słownik{$_}\n";  
}
```

- ▶ Funkcja wbudowana `keys` zwraca listę wszystkich kluczy danego hasza
- ▶ Można tę listę zapisać do zmiennej tablicowej lub użyć w dowolnym kontekście listowym
- ▶ Elementy w haszach nie są uporządkowane, więc lista zwrócona przez `keys` też nie jest

Przeglądanie haszów - według wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (values %słownik) {  
    print "Słownik zawiera nast. części mowy: \n";  
}
```

Przeglądanie haszów - według wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
5  
foreach (values %słownik) {  
    print "Słownik zawiera nast. części mowy: \n";  
}
```

- ▶ Funkcja wbudowana `values` zwraca listę wszystkich wartości danego hasza (odpowiednik `keys`)

Przeglądanie haszów - według wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5 foreach (values %słownik) {  
    print "Słownik zawiera nast. części mowy: \n";  
}
```

- ▶ Funkcja wbudowana `values` zwraca listę wszystkich wartości danego hasza (odpowiednik `keys`)
- ▶ W przeciwieństwie do kluczy, wartości w haszu mogą się powtarzać (hasze są lewostronnie jednoznaczne)

Przeglądanie haszów - według wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
5  
foreach (values %słownik) {  
    print "Słownik zawiera nast. części mowy: \n";  
}
```

- ▶ Funkcja wbudowana `values` zwraca listę wszystkich wartości danego hasza (odpowiednik `keys`)
- ▶ W przeciwieństwie do kluczy, wartości w haszu mogą się powtarzać (hasze są lewostronnie jednoznaczne)
- ▶ Nie ma bezpośredniej możliwości wyświetlenia klucza do odpowiedniej wartości

Przeglądanie haszów - według par kluczy i wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5  
while(my ($wyraz, $czesc_mowy) = each %słownik) {  
    print "Wyraz $wyraz to $czesc_mowy\n";  
}
```

Przeglądanie haszów - według par kluczy i wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5  
while(my ($wyraz, $czesc_mowy) = each %słownik) {  
    print "Wyraz $wyraz to $czesc_mowy\n";  
}
```

- ▶ Funkcja each w kontekście listowym dla podanego hasza zwraca parę klucz-wartość (czyli listę dwuelementową)

Przeglądanie haszów - według par kluczy i wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5  
while(my ($wyraz, $czesc_mowy) = each %słownik) {  
    print "Wyraz $wyraz to $czesc_mowy\n";  
}
```

- ▶ Funkcja each w kontekście listowym dla podanego hasza zwraca parę klucz-wartość (czyli listę dwuelementową)
- ▶ Przy każdym wywołaniu each zwraca kolejną parę, aż zabraknie par w haszu

Przeglądanie haszów - według par kluczy i wartości

```
1 my %słownik = (  
    szafa => "rzeczownik", wielki => "przymiotnik",  
    mrugac => "czasownik", krotko => "przysłówek",  
);  
  
5  
while(my ($wyraz, $czesc_mowy) = each %słownik) {  
    print "Wyraz $wyraz to $czesc_mowy\n";  
}
```

- ▶ Funkcja each w kontekście listowym dla podanego hasza zwraca parę klucz-wartość (czyli listę dwuelementową)
- ▶ Przy każdym wywołaniu each zwraca kolejną parę, aż zabraknie par w haszu
- ▶ W kontekście skalarnym zwraca jedynie kolejne klucze

- ▶ Hasze to struktury podobne do list, które mają zawsze parzystą liczbę elementów (możemy zapisywać tablice do haszów i hasze do tablic)

Hasze - podsumowanie

- ▶ Hasze to struktury podobne do list, które mają zawsze parzystą liczbę elementów (możemy zapisywać tablice do haszów i hasze do tablic)
- ▶ Hasze można traktować jak zbiory par klucz-wartość (zbiór par uporządkowanych, relacja)

- ▶ Hasze to struktury podobne do list, które mają zawsze parzystą liczbę elementów (możemy zapisywać tablice do haszów i hasze do tablic)
- ▶ Hasze można traktować jak zbiory par klucz-wartość (zbiór par uporządkowanych, relacja)
- ▶ Elementy haszów nie są uporządkowane (tak jak zbiór)

- ▶ Hasze to struktury podobne do list, które mają zawsze parzystą liczbę elementów (możemy zapisywać tablice do haszów i hasze do tablic)
- ▶ Hasze można traktować jak zbiory par klucz-wartość (zbiór par uporządkowanych, relacja)
- ▶ Elementy haszów nie są uporządkowane (tak jak zbiór)
- ▶ Klucze haszy nie mogą się powtarzać, próba dodania pary klucz-wartość dla istniejącego klucza spowodują nadpisanie poprzedniej wartości (relacja lewostronnie jednoznaczna)

- ▶ Hasze to struktury podobne do list, które mają zawsze parzystą liczbę elementów (możemy zapisywać tablice do haszów i hasze do tablic)
- ▶ Hasze można traktować jak zbiory par klucz-wartość (zbiór par uporządkowanych, relacja)
- ▶ Elementy haszów nie są uporządkowane (tak jak zbiór)
- ▶ Klucze haszy nie mogą się powtarzać, próba dodania pary klucz-wartość dla istniejącego klucza spowodują nadpisanie poprzedniej wartości (relacja lewostronnie jednoznaczna)
- ▶ Operator `=>` jest synonimem przecinka `,` (operator listowy), dodatkowo wymusza po lewej stronie kontekst łańcuchowy (nawet gdy klucz jest liczbą!)

Jeśli chcemy uporządkować tablicę według jakiegoś określonego porządku korzystamy z funkcji `sort`

```
1 @lista_obecnosci = qw(Zenon Wladek Antek Mirek Edek);  
  print join("\n",sort @lista_obecnosci)."\n";
```

Jeśli chcemy uporządkować tablicę według jakiegoś określonego porządku korzystamy z funkcji `sort`

```
1 @lista_obecnosci = qw(Zenon Wladek Antek Mirek Edek);  
  print join("\n",sort @lista_obecnosci).\n";
```

Sortowanie działa też na liczbach

```
1 @liczby = (4,7,13,9,5,2,10,7);  
  print join(", ",sort @liczby).\n";
```

Jeśli chcemy uporządkować tablicę według jakiegoś określonego porządku korzystamy z funkcji `sort`

```
1 @lista_obecnosci = qw(Zenon Wladek Antek Mirek Edek);  
  print join("\n", sort @lista_obecnosci). "\n";
```

Sortowanie działa też na liczbach

```
1 @liczby = (4,7,13,9,5,2,10,7);  
  print join(", ", sort @liczby). "\n";
```

Ale może działać dziwnie dla wartości mieszanych

```
1 @mieszane = (4, "Antek", 13, 9, "Zenon", 2, 10, "Mirek");  
  print join(", ", sort @mieszane). "\n";
```

Sortowanie tablic

Jeśli chcemy uporządkować tablicę według jakiegoś określonego porządku korzystamy z funkcji `sort`

```
1 @lista_obecnosci = qw(Zenon Wladek Antek Mirek Edek);  
  print join("\n", sort @lista_obecnosci). "\n";
```

Sortowanie działa też na liczbach

```
1 @liczby = (4,7,13,9,5,2,10,7);  
  print join(", ", sort @liczby). "\n";
```

Ale może działać dziwnie dla wartości mieszanych

```
1 @mieszane = (4, "Antek", 13, 9, "Zenon", 2, 10, "Mirek");  
  print join(", ", sort @mieszane). "\n";
```

Według jakiego porządku została posortowana ostatnia lista?

Sortowanie tablic - ciąg dalszy

Funkcja `sort` może działać według dowolnych porządków

```
1 @mieszane = (4, "Antek", 13, 9, "Zenon", 2, 10, "Mirek");  
print join(",", sort {  
    $a <=> $b or $a cmp $b  
} @mieszane)."\n";
```

Kto potrafi wytłumaczyć, dlaczego taki zapis porządkuje w obserwowany sposób?

Wskazówka: Operator logiczny `or` nie sprawdza wyrażenia po jego prawej stronie, gdy wyrażenie po lewej stronie jest prawdziwe

Funkcja `sort` może działać według dowolnych porządków

```
1 @mieszane = (4, "Antek", 13, 9, "Zenon", 2, 10, "Mirek");  
print join(",", sort {  
    $a <=> $b or $a cmp $b  
} @mieszane). "\n";
```

Kto potrafi wytłumaczyć, dlaczego taki zapis porządkuje w obserwowany sposób?

Wskazówka: Operator logiczny `or` nie sprawdza wyrażenia po jego prawej stronie, gdy wyrażenie po lewej stronie jest prawdziwe

- ▶ Zmienne `$a` i `$b` reprezentują dwie porównywane wartości sortowanej listy
- ▶ Określając sposoby porównywania, określamy porządki sortowania
- ▶ Sortowanie, gdzie liczby poprzedzają łańcuchy jest trudniejsze

Inne sortowania

```
1 @lista = qw(Waldek Zenek tort Tomek Olga Ala worek);

print join(",", sort @lista). "\n";
print join(",", sort {lc($a) cmp lc($b)} @lista). "\n";
5 print join(",", sort {lc($b) cmp lc($a)} @lista). "\n";

@revlista = reverse sort {lc($a) cmp lc($b)} @lista;
print join(",", @revlista). "\n";

10 print join(",", sort {
    length($a) <=> length($b) or $a cmp $b
} @lista). "\n";

print join(",", sort {
15     reverse($a) cmp reverse($b)
} @lista). "\n";
```


Sortowanie a hasze

```
1 my %hasz = (  
    bb => "zz", aa => "yy", 11 => "xx",  
);  
  
5 print "$_ => $hasz{$_}\n"  
    foreach (sort mysort keys %hasz);  
  
sub mysort {  
    return $a <=> $b or $a cmp $b;  
10 }
```

Sortowanie a hasze

```
1 my %hasz = (  
    bb => "zz", aa => "yy", ll => "xx",  
);  
  
5 print "$_ => $hasz{$_}\n"  
    foreach (sort mysort keys %hasz);  
  
sub mysort {  
    return $a <=> $b or $a cmp $b;  
10 }
```

- ▶ Hasze nie mają określonego porządku
- ▶ Za pomocą funkcji `sort` oraz np. `keys` możemy sobie sami określić taki porządek
- ▶ Kryteria sortowania można określić we własnej funkcji, zmienne `$a` i `$b` są standardowo dostępne

Funkcja map - Funkcyjne przetwarzanie tablic

```
1 map { BLOK } LISTA  
   map( WYRAŻENIE, LISTA )
```

Funkcja map - Funkcyjne przetwarzanie tablic

```
1 map { BLOK } LISTA  
map( WYRAŻENIE, LISTA )
```

- ▶ Oblicza wartość w bloku lub wyrażeniu dla każdego elementu tablicy (iterowanych za pomocą \$_)

Funkcja map - Funkcyjne przetwarzanie tablic

```
1 map { BLOK } LISTA  
map( WYRAŻENIE, LISTA )
```

- ▶ Oblicza wartość w bloku lub wyrażeniu dla każdego elementu tablicy (iterowanych za pomocą \$_)
- ▶ Zwraca listę elementów powstałych przez takie obliczenie

Funkcja map - Funkcyjne przetwarzanie tablic

```
1 map { BLOK } LISTA  
map( WYRAŻENIE , LISTA )
```

- ▶ Oblicza wartość w bloku lub wyrażeniu dla każdego elementu tablicy (iterowanych za pomocą \$_)
- ▶ Zwraca listę elementów powstałych przez takie obliczenie
- ▶ W kontekście skalarnym zwraca liczbę elementów tak wygenerowanych

Funkcja map - Funkcyjne przetwarzanie tablic

```
1 map { BLOK } LISTA  
map( WYRAŻENIE , LISTA )
```

- ▶ Oblicza wartość w bloku lub wyrażeniu dla każdego elementu tablicy (iterowanych za pomocą \$_)
- ▶ Zwraca listę elementów powstałych przez takie obliczenie
- ▶ W kontekście skalarnym zwraca liczbę elementów tak wygenerowanych
- ▶ Blok lub wyrażenie są obliczane w kontekście skalarnym, mogą więc zwrócić zero, jeden lub kilka elementów

Przykład – generowanie listy odmiany przymiotników

```
1 sub generuj {  
    my $adj = shift;  
    return map {  
        "$adj$_", "${adj}er$_", "${adj}st$_"  
5    } qw(er e es en em);  
}
```


Przykład – generowanie listy odmiany przymiotników

```
1 sub generuj {  
    my $adj = shift;  
    return map {  
        "$adj$_", "${adj}er$_", "${adj}st$_"  
5    } qw(er e es en em);  
}
```

- ▶ Funkcja `map` tworzy dla każdej końcówki fleksyjnej trzelementową listę (rdzenia przymiotnika w stopniu równym, wyższym, najwyższym)

Przykład – generowanie listy odmiany przymiotników

```
1 sub generuj {  
    my $adj = shift;  
    return map {  
        "$adj$_", "${adj}er$_", "${adj}st$_"  
5    } qw(er e es en em);  
}
```

- ▶ Funkcja `map` tworzy dla każdej końcówki fleksyjnej trzelementową listę (rdzenia przymiotnika w stopniu równym, wyższym, najwyższym)
- ▶ Wynikiem jest lista 15-elementowa (3×5) – pamiętamy, że dwie listy w kontekście listowym łączą się w jedną większą listę

Przykład – generowanie listy odmiany przymiotników

```
1 sub generuj {  
    my $adj = shift;  
    return map {  
        "$adj$_", "${adj}er$_", "${adj}st$_"  
5    } qw(er e es en em);  
}
```

- ▶ Funkcja `map` tworzy dla każdej końcówki fleksyjnej trzelementową listę (rdzenia przymiotnika w stopniu równym, wyższym, najwyższym)
- ▶ Wynikiem jest lista 15-elementowa (3×5) – pamiętamy, że dwie listy w kontekście listowym łączą się w jedną większą listę
- ▶ Lista jest zwracana za pomocą `return`, nie ma potrzeby tworzenia tablicy tymczasowej

Przykład – usuwanie powtarzających się elementów

```
1 my @tablica = qw(aa ab bb aa bb ac ab aa);  
  
my %hasz = map { $_ => 1 } @tablica;  
@pojedyncze = sort keys %hasz;  
5  
print join(" ", @pojedyncze)."\n";
```

Przykład – usuwanie powtarzających się elementów

```
1 my @tablica = qw(aa ab bb aa bb ac ab aa);  
  
my %hasz = map { $_ => 1 } @tablica;  
@pojedyncze = sort keys %hasz;  
5  
print join(" ", @pojedyncze)."\n";
```

- ▶ Funkcja `map` tworzy listę, w której nieparzyste elementy pochodzą z `@tablica`, parzyste elementy to `1`

Przykład – usuwanie powtarzających się elementów

```
1 my @tablica = qw(aa ab bb aa bb ac ab aa);  
  
my %hasz = map { $_ => 1 } @tablica;  
@pojedyncze = sort keys %hasz;  
5  
print join(" ", @pojedyncze)."\n";
```

- ▶ Funkcja `map` tworzy listę, w której nieparzyste elementy pochodzą z `@tablica`, parzyste elementy to `1`
- ▶ Przypisanie tej listy do hasza zamienia nieparzyste elementy na klucze, parzyste na wartości hasza

Przykład – usuwanie powtarzających się elementów

```
1 my @tablica = qw(aa ab bb aa bb ac ab aa);  
  
my %hasz = map { $_ => 1 } @tablica;  
@pojedyncze = sort keys %hasz;  
5  
print join(" ", @pojedyncze)."\n";
```

- ▶ Funkcja `map` tworzy listę, w której nieparzyste elementy pochodzą z `@tablica`, parzyste elementy to `1`
- ▶ Przypisanie tej listy do hasza zamienia nieparzyste elementy na klucze, parzyste na wartości hasza
- ▶ W haszu klucze nie mogą się powtarzać (wartości skojarzone z istniejącym kluczem zostaną nadpisane przez nową wartość)

Przykład – usuwanie powtarzających się elementów

```
1 my @tablica = qw(aa ab bb aa bb ac ab aa);  
  
my %hasz = map { $_ => 1 } @tablica;  
@pojedyncze = sort keys %hasz;  
5  
print join(" ", @pojedyncze)."\n";
```

- ▶ Funkcja `map` tworzy listę, w której nieparzyste elementy pochodzą z `@tablica`, parzyste elementy to `1`
- ▶ Przypisanie tej listy do hasza zamienia nieparzyste elementy na klucze, parzyste na wartości hasza
- ▶ W haszu klucze nie mogą się powtarzać (wartości skojarzone z istniejącym kluczem zostaną nadpisane przez nową wartość)
- ▶ Tablica składająca się z samych kluczy tego hasza jest tablicą zawierającą tylko niepowtarzające się elementy z `@tablica`

- ▶ Hasze to jeden z najważniejszych mechanizmów w Perlu

- ▶ Hasze to jeden z najważniejszych mechanizmów w Perlu
- ▶ Bardzo wiele zadań programistycznych można rozwiązać w elegancki sposób za pomocą haszy (zadania domowe)

- ▶ Hasze to jeden z najważniejszych mechanizmów w Perlu
- ▶ Bardzo wiele zadań programistycznych można rozwiązać w elegancki sposób za pomocą haszy (zadania domowe)
- ▶ Poznaliśmy kilka funkcyjnych sposobów przetwarzania list i haszów (wejściem do funkcji jest lista, wyjściem lista przetworzona)

- ▶ Hasze to jeden z najważniejszych mechanizmów w Perlu
- ▶ Bardzo wiele zadań programistycznych można rozwiązać w elegancki sposób za pomocą haszy (zadania domowe)
- ▶ Poznaliśmy kilka funkcyjnych sposobów przetwarzania list i haszów (wejściem do funkcji jest lista, wyjściem lista przetworzona)
- ▶ Między innymi dzięki tym funkcjom składnia Perla jest taka zwięzła

- ▶ Hasze to jeden z najważniejszych mechanizmów w Perlu
- ▶ Bardzo wiele zadań programistycznych można rozwiązać w elegancki sposób za pomocą haszy (zadania domowe)
- ▶ Poznaliśmy kilka funkcyjnych sposobów przetwarzania list i haszów (wejściem do funkcji jest lista, wyjściem lista przetworzona)
- ▶ Między innymi dzięki tym funkcjom składnia Perla jest taka zwięzła
- ▶ Zadania wykonywane przez te funkcje zajęłyby kilka wierszy każdym tradycyjnym języku programowania