

## Referenciák

1

## Referenciák

- Perl5 legfontosabb újdonsága volt
- Nagyon hasznos típus, lehetővé teszi bonyolultabb adatszerkezetek kezelését
- Speciális skalár típus
- Mutathat más skalárra, listára, hash-re
- Mutathatunk vele egy eljárásra, anonymous szubrutinra
- Mutathat glob-ra
- Mutathat balértékre (például egy substr visszatérési értéke)
- személggyűjtés

2

## Skalárra mutató referenciák

- A \ operátorral állíthatunk referenciát egy skalárra
- Hasznos lehet, ha
  - nem akarunk egy nagy sztringet átadni egy alprogramnak
  - az esetleg módosítani akarja
  - referenciára van szükségünk

3

## Skalárra mutató referenciák

- ```
my $html_source = q{
<!DOCTYPE HTML PUBLIC ...>
<html>
...
};
```
- ```
my $html_ref = \$html_source;
```
- Hivatkozott skalár elérése:
  - `$$html_ref`
  - `${html_ref}`

4

## Listareferenciák

- Listákra mutató referenciák többféleképp hozhatók létre
- Egy meglévő listára a \ operátorral állíthatunk referenciát
- `$list_ref = \@list;`
- Amíg hivatkozás van a listára, addig nem szűnik meg
- Visszaadhatunk listára mutató referenciát egy eljárásból

5

## Listareferenciák

- ```
my @list = (10..20);
```
- ```
my $ref = \@list;
```
- ```
my $other_ref = [ 10, 12, 20, 12 ];
```
- a [ ] operátor új listát készít, és arra ad vissza referenciát
- Leírhatunk ilyet is:
  - ```
@AoA = (
  [ 7, 8, 9 ],
  [ 4, 5, 6 ],
  [ 1, 2, 3 ],
);
```

6

## Listareferenciák

- számológép gombjai
- `my @upper = (7, 8, 9);`
- `my @middle = (4, 5, 6);`
- `my @bottom = (1, 2, 3);`
- `my @buttons = (@upper, @middle, @bottom);`
  - ez nem megfelelő, mert a Perl a listákat "kiegyenesíti"
  - az eredmény:  
`(7, 8, 9, 4, 5, 6, 1, 2, 3);`
- `my @buttons = (\@upper, \@middle, \@bottom);`
- `my @buttons = \@upper, @middle, @bottom;`
  - ugyanaz(!), mint az előző
  - nem pedig ez:  
`[ 7, 8, 9, 4, 5, 6, 1, 2, 3];`

7

## Listaelemek hivatkozása

- `my @list = (10..20);`
- `my $list = \@list;`
- nem referencián keresztül:
  - `$list[0]`
- referencián keresztül
  - `$$list[0]`
    - jelentése: `$$list[0]`
    - az indexelés csak a dereferálás után következik
  - `$list->[0]`
    - olvashatóbb szintaxis
    - látszik, hogy mi tartalmazza a referenciát

8

## Többdimenziós tömbök

- `$ref_AoA = [ [7, 8, 9], [4, 5, 6], [1, 2, 3], ];`
- `$$ref_AoA[2][1];`
  - Not a SCALAR reference ...
- `$$ref_AoA[2][1];`
  - zavaros, nehezen olvasható
- `$ref_AoA->[2]->[1];`
  - sokkal tisztább, egyértelmű
- syntactic sugar:
  - `$ref_AoA->[2][1];`
  - `$AoA[2][1];`

9

## Lista hivatkozása

- Nem csak egy listaelemre lehet szükségünk, hanem az egész listára is (például `push`)
- Az egész lista:
  - `@$ref`
  - `@{$ref}`
  - `@{ $hash_of_lists{'foo'} }`
  - `@{$ref_AoA->[0]}`
- `push(@$list_ref, 'foo');`
- `{}` között tetszőleges kifejezés is szerepelhet

10

## Hash referenciák

- Meglévő hash-re a `\` operátorral állítható referencia
- `my %hash = ( 'a' => 10, 'b' => 11, 'c' => 12, 'd' => 13, 'e' => 14, 'f' => 15, );`
- `my $hash_ref = \%hash;`
- A hash nem szabadul fel, amíg mutat rá referencia
- `my $hash_ref = { 'a' => 10, 'b' => 11, 'c' => 12, 'd' => 13, 'e' => 14, 'f' => 15, };`

11

## Hash referenciák

- `my $empty_hash_ref = {};`
- `my $hash_ref = { 'odds' => [ 1, 3, 5, 7 ], 'evens' => [ 0, 2, 4, 6 ], };`
- `my $room = { 'bed' => { color => 'red/brown', length => '220', }, table => { color => 'red', }, };`

12

## Hash referenciák: értékek elérése

- nem referencia esetében:
  - `$hash{'a'}; # 10`
- referencia esetében
  - `$$hash{'a'}`
  - jelentése `#{hash}{'a'}`
  - de nem `#{hash{'a'}}`
- olvashatóbb szintaxis
  - `$hash->{'a'}`
- fenti példa:
  - Milyen színű az ágy?
  - `#{room{'bed'}}{'color'}`
  - `$room->{'bed'}->{'color'}`
  - syntactic sugar:
    - `$room->{'bed'}{'color'}`

13

## Hivatkozott hash elérése

- Természetesen a teljes hivatkozott hash is elérhető
- `#{hash_ref}`
- `#{hash_ref}`
- `#{room->{'bed'}}`
- Kírní mindent az ágyról:
- `foreach my $attr (keys %{room->{'bed'}}) {  
 print "$attr: $room->{'bed'}{$attr}\n";  
}`

14

## Szubrutin referencia

subref.pl  
subref-modify.pl

- A `\` operátorral egy alprogramra is állíthatunk referenciát
- `my $subref = \mysub;`
- a hivatkozott alprogram meghívása
  - `&$subref;`
  - `&$subref();`
- paraméterek átadása
  - `&$subref($arg);`
- más szintaxis
  - `$subref->();`
  - `$subref->($arg);`
- probléma: figyelmen kívül hagyja a prototípusokat

15

## Anonymous alprogramok

anonymous\_sub.pl

- `my $anonymous_sub = sub {  
 print "Hello world\n";  
};`
- `$anonymous_sub->();`
- `&$anonymous_sub;`
- Egy ilyen referenciát változóban tárolunk, ezért átadhatjuk paraméterül, lehet visszatérési érték, stb.

16

## Closure

- Egy ilyen sub a környezetét is megjegyzi
  - closure
- ```
my $sub;  
{  
  my @value = (10, 11);  
  $sub = sub { return @value; }  
}
```
- `print $sub->(); # 1011`
- a tömb nem szabadul fel, mert az anonymous sub hivatkozik rá

17

## Closure

- lexikálisan már nem látható az `@value` változóban tárolt tömb, de az élettartama túlnyúlik
- ha el akarjuk érni, kell rá állítani egy referenciát
- ```
my ($sub, $ref);  
{  
  my @value = (10, 11);  
  $ref = \@value;  
  $sub = sub { return @value; }  
}
```
- `print $sub->(); # 1011`
- `$ref->[0] = 20;`
- `print $sub->(); # 2011`

18

## Szimbolikus referenciák

- Az eddig tárgyalt referenciákat 'hard reference' néven említik
- 'soft reference' – ne használjuk
- 'symbolic reference'
- egy változóra a nevével hivatkozhatunk
- `$value = 10;`  
`$name = 'value';`  
`$$name; # 10`  
`$$name = 11; # $value == 11`
- `use strict;`
  - a pragma nem engedélyezi ezek használatát

19

## Szimbolikus referenciák

- Bonyolultabb kifejezés is írható:
- `${$name} = 10;`
  - ugyanaz, mint az előző, csak explicitebb szintaxissal
- `$value_1 = 100;`  
`${$name.'_1'}; # 100`
- Listákra, hash-ekre is működik
- `@list = (1,2,3);`  
`$name = 'list';`  
`push(@$name, 4);`  
`@list: (1,2,3,4)`

20

## Szimbolikus referenciák

- Függvényeket is hívhatunk így:  

```
sub hello {  
    print "Hello world!\n";  
}  
$name = 'hello';  
&$name(); # Hello world!
```
- Hasznos, ha egy csomag nevét és egy benne lévő metódus nevét tudjuk, vagy a metódus nevét dinamikusan tudjuk meghatározni
- ```
sub hello_pete {  
    print "Hello Pete!\n";  
}  
$who = 'pete';  
&{'hello_' . $who};
```

21

## Szimbolikus referenciák

- `use strict 'refs';`
- `use strict qw(refs);`
- Így nem használhatóak a szimbolikus referenciák
- Nem véletlenül
- Ha mégis szükségünk van rá, akkor kikapcsoljuk ezt az ellenőrzést egy blokkban
- ```
{  
    no strict 'refs';  
    ...  
}
```

22

## I/O műveletek

23

## I/O operátorok

- filehandle segítségével végezhetünk I/O műveleteket
- a filehandle neve csupa nagybetű
  - nem kötelező, csak konvenció
- nincs külön karakter a név előtt (mint pl. \$, %, @)
  - csupa kisbetű azonosítókat megkülönböztető jel (\$, @, %, \*) nélkül a Perl fenntartja potenciális jövőbeni foglalt kulcsszónak (warning)
- előre definiált filehandle: STDIN, STDERR, STDOUT
  - ha van `__DATA__` sor a file-ban, akkor DATA is

24

## FILEHANDLE-re írás

- egy filehandle-re írhatunk
  - print FILEHANDLE \$string;
- print STDERR "Debug: \ \$x=\$x\n";
  - zárójel tetszőleges helyre tehető
  - print(STDERR "Debug: \ \$x=\$x\n");
  - print STDERR ("Debug: \ \$x=\$x\n");
  - de általában inkább nem tesszük ki
- print "...";
  - ekvivalens ezzel:
  - print STDOUT "...";

25

## FILEHANDLE-ből olvasás

- olvasni a <> operátorral lehet
- <FILEHANDLE>
  - skalár környezetben egy sort ad vissza
  - újsor karaktert is beolvassa (ha van)
- print "Delete directory (y/n) ? ";  
my \$yes\_or\_no = <STDIN>;  
chomp(\$yes\_or\_no);
- while (my \$line = <STDIN>) {  
  print \$line;  
}

26

## FILEHANDLE-ből olvasás

filehandle-read.pl  
filehandle-read-all.pl

- <FILEHANDLE>
  - lista környezetben egy listát ad vissza
  - minden sort, amíg a FILEHANDLE-ről EOF nem érkezik
- my @lines = <STDIN>;  
print STDERR scalar(@lines) . ' lines read';  
print @lines;
  - minden sor egy külön sztring a listában
  - minden sort újsor karakter zár le

27

## File-ok megnyitása

- open függvény
  - shell style
- sysopen függvény
  - C style
- open nagyon sokoldalú
  - szöveges/bináris file-ok olvasása
  - írhatunk változóba egy megfelelő open hívással egy filehandle-n keresztül
  - pipe-ok
  - alprocesszek nyitása
  - stb.

28

## File-ok megnyitása

- open(HANDLE, 'file.txt')  
  or die "Can't open file.txt for reading: \$!\n";
  - ha a HANDLE már meg volt nyitva, akkor először a Perl lezárja, aztán nyitja meg újra
  - open visszatérési értéke hamis, ha hiba volt
  - hiba esetén a \$! változó tartalmazza az operációs rendszertől kapott hibaüzenetet
- open(HANDLE, '<file.txt')  
  or die "Can't open file.txt for reading: \$!\n";
- open(HANDLE, '>file.txt')  
  or die "Can't open file.txt for writing: \$!\n";
- open(HANDLE, '>>file.txt')  
  or die "Can't open file.txt for append: \$!\n";

29

## File-ok megnyitása

- filenév elején és végén található fehér szóközök (white space) nem számítanak
  - open(HANDLE, '>file.txt') or die;
  - open(HANDLE, '> file.txt') or die;
  - open(HANDLE, '> ' file.txt ') or die;
- ez a tulajdonság segít abban, hogy a következő jól működhessen akkor is, ha nem vágjuk a le a sor vége jelet
- ...  
\$filename = <INFO>;  
open(EXTRA, "< \$filename")  
  or die "Can't open \$filename: \$!\n";

30

## File-ok lezárása

- close
- open implicit lezárja a HANDLE-t
- Perl kilépéskor bezárja a nyitott file-okat
- explicit kitenni jó
  - close HANDLE;
  - close(HANDLE);
- pipe esetében megvárja a másik processz befejeződését
- visszatérési értéke van
  - hamis, ha nem sikerült lezárni
    - pl. nem sikerült minden I/O puffert megfelelően üríteni
    - pipe-ok esetében a másik processz problémái
  - close(HANDLE) or warn "Close failed: \$!\n";

31

## File handle

- Egy handle-t nem kell előre definiálni use strict; esetén sem
  - lehetőség sincs rá
  - szintaktikai hiba:
    - my HANDLE;
- a scope-ja dinamikus lesz
- meg nem nyitott HANDLE-ből történő olvasás nem végzetes hiba, csak warning

32

## File handle paraméterek

- nincs kezdő karakter, ami jelezné, hogy ez az azonosító egy filehandle
- egy handle paraméterül átadása nehezebb
  - a handle neve csak néhány helyen szabályos és értelmes
    - open
    - close
    - <>
    - print
  - egyébként bareword
- egy filehandle-t ezért nem tudunk átadni paraméterként
- workaround:
  - type glob

33

## Type glob

- \$alma, @alma, %alma külön változók
  - emellett lehet még egy alma nevű filehandle is
- \* karakter használható egy adott névhez tartozó összes változó jelölésére
- \*alias = \*variable;
- ennek segítségével alias-okat hozhatok létre
  - nincs másolás, csak a szimbólumtábla módosul!
  - csak azokra a típusokra, amihez az eredeti is létezik
- filehandle-k átadására használható
- \*STDERR = \*STDOUT;
  - print STDERR "error\n";
  - print "output\n";

34

## File handle local

- local segítségével újra megnyithatom egy adott blokkban
  - erre a feladatra csak local használható
  - my nem segíthet és nem is segítene
- a már megnyitott file nyitva marad
- a blokkból való kilépés után minden folytatódik tovább az előző handle-vel

35

## Pipe-ok

- Unix shellben használhatunk pipe-okat, hogy az egyik parancs kimenetét a másik bemenetére irányítsuk
  - dmesg | less
  - find . -type f | xargs chmod -x
  - stb.
- Hasonló szintaxissal nyithatunk meg pipe-okat Perlben is
- open
  - open(PIPEHANDLE, '| cmd')
  - or die "Can't open pipe: \$!\n";
  - open(PIPEHANDLE, 'cmd |')
  - or die "Can't open pipe: \$!\n";

36

## Bináris fileok kezelése

- binmode függvénnyel lehet
  - olyan operációs rendszereken, amely megkülönböztet szöveges és bináris fileokat
  - pl. DOS/Windows szövegfileokban a \cZ karakter a file végét jelenti: binmode kell
  - implicit CRLF->LF transzformáció kiküszöbölése
    - Perl olvasáskor a CRLF karaktereket egy szövegfileban \n karakterekre cseréli (OS-függő)
    - íráskor pedig a \n-t CRLF-re cseréli (OS-függő)

37

## Bináris file-ok kezelése

- ```
open(ORIG, "original");
open(COPY, ">copy");
binmode(ORIG);
binmode(COPY);
while (read(ORIG, $buffer, 1024) {
    syswrite(COPY, $buffer);
}
close(COPY);
close(ORIG);
```

38

## sysopen

- open:
  - pufferelt filehandle-k
- sysopen:
  - nem pufferelt
- precízebb beállításokat tesz lehetővé
  - open(HANDLE, ">output") or die;
  - mindig felülírja a filelet
  - (persze a létezése le is ellenőrizhető)
  - O\_EXCL opcióval a sysopen ezt nem teszi meg
- Példa:
  - sysopen(FH, \$path,
  - O\_WRONLY | O\_TRUNC | O\_CREAT);
  - ekvivalens:
  - open(FH, "> \$path");

39

filehandle.pl

## Filekezelés máshogy

- Használhatunk objektum-orientált interfészt is
  - IO::File
  - FileHandle
- előnye, hogy az objektumok skalárok (igaz, speciális tulajdonsággal)
- átadhatóak paraméterként minden trükk nélkül
- lehetnek lexikálisak

40

dirhandle.pl

## Könyvtárak kezelése

- DIRHANDLE használható könyvtárak tartalmának olvasására
  - DIRHANDLE hasonló a FILEHANDLE-hez
  - saját névterük van
  - typeglob-ok ezeket is tartalmazzák
- opendir
- readdir
- closedir
- File::Find modul egy rekurzív megfelelője ennek a funkcionalitásnak
  - find program

41

example.pl

## A DATA filehandle

- egy előre megnyitott filehandle
- Perl nyitvahagyja a forráskódot olvasó filehandle-t
- forráskódban jelezni kell egy külön sorral
  - \_\_DATA\_\_
  - ami ezután jön, a Perl nem értelmezi
- DATA handle
  - közöséges filehandle
  - az ezután következő sorok olvashatók ki a handle-ből
  - dolgunk végeztével le kell zárni

42

## Kivételek

43

## Dinamikus kód futtatás

- futási időben meghatározott kódrészlet futtatása
- Perl a Perlben
- az interpreter teszi ezt lehetővé
- `eval('print "Hello world\n");`
- `eval(...);`
  - az eredményt az `eval` visszaadja
  - ha valami hiba lép fel, akkor a visszatérési érték `undef`, a hiba pedig elérhető lesz a `$@` változóban
  - természetesen a kód fordítási időben történő ellenőrzésére nincs mód
  - minden híváskor fordítás

44

## Dinamikus kód futtatás

eval-op.pl

- `eval` eredménye a blokk eredménye
  - utolsó kiértékelt kifejezés értéke
  - `return` által visszaadott érték
- `my $quotient = eval('$a/$b');`  
`$quotient = 0 if $@;`
- `my $quotient;`  
`eval('$quotient = $a/$b');`  
`$quotient = 0 if $@;`

45

## Dinamikus kód futtatás

- `eval` blokkokat is használhatunk
  - fordítási időben ellenőrzött
  - fordítási időben fordított
- `eval {`
  - ...
  - `};`
  - `my $quotient = eval {`
    - `$a / $b`
    - `};`
    - `$quotient = 0 if $@;`

46

## die

- A programból bármikor kiléphetünk az `exit` függvénnyel
  - nem kezelhető ez a kilépés egy felsőbb szintről
- `die`
  - kilépés a programból
  - nem `exit code`-ot adunk meg, hanem üzenetet

47

## die

- `die "Not a number" unless $numeric =~ /\^d+$/;`
  - at <file> line <line>
  - Not a number at test.pl line 7.
- `die "Not a number\n"`
  - unless `$numeric =~ /\^d+$/;`
  - nem fűz hozzá semmit
  - Not a number
- `die;`
  - Died at ...
  - Died at humbug.pl line 13.

48



## Kivételek

- egy ilyen hiba nem lép ki azonnal
- eval blokkba ágyazva nem lép ki a program
  - hiba esetén az eval undef-et ad vissza
  - hiba esetén a hiba a \$@ változóban van (ami egyébként "")
  - a hiba tovább terjed a hívási veremben
- a hibát nem kell lekezelnem, csak ha akarom
  - eval { ... };
  - a hiba ilyenkor csendes lesz, nem terjed tovább

49

exceptions.pl

## Kivételkezelés összehasonlítása

- |           |                          |
|-----------|--------------------------|
| • throw   | • die                    |
| • catch   | • if (\$@) {...}         |
| • finally | • blokk utáni utasítások |
| • else    | • unless (\$@) {...}     |

50

## Ami dobható

- skalár dobható
  - szöveg/szám
  - referencia is
  - objektum is
- ha tömböt vagy hash-t dobunk
  - nem hiba
  - de „sztringgé konvertálódnak”, és a kivétel már egyszerű skalár lesz
- ha elkapom leellenőrizhetem
  - \$@ változó vizsgálata

51

## Scripting

52

## Scripting

- elég sokmindent átvettünk már
- alapvető dolgok hiányoznak még ahhoz, hogy komolyabb scripteket írassunk
- ezek közül néhány tárgyalása következik

53

## warn

- warn “Failed to close”;
  - Failed to close at test.pl line 6.
- warn “Failed to close\n”;
  - Failed to close
- STDERR-re ír
- nem hiba, csak warning

54

## warning

- A Perl fordító figyelmeztethet néhány hibára
  - ha csak egyszer használunk egy változót
  - stb.
- futási idejű figyelmeztetés
  - ha nem megfelelően használunk egy változót
  - inicializálatlan érték használata (undef)
  - nem szám érték használata rossz helyen
  - stb.
- -w kapcsoló mindig javasolt
  - warning ne maradjon a programban
  - -w kapcsoló az egész programra érvényes
  - a \$^W globális változóval befolyásolhatjuk a működését

55

warning.pl

## diagnostics

- Ez a pragma bőbeszédű hibaüzeneteket és figyelmeztetéseket eredményez
- kezdőknek nagyon hasznos, mert példákkal próbál rávezetni arra, hogy mi lehet a hiba
- use diagnostics;

56

warnings-pragma.pl

## use warnings;

- Perl 5.6.0 verziótól kezdve
- warnings pragma a figyelmeztetések kezelésének modern módja
- előnyei:
  - lexikálisan kapcsolhatjuk a figyelmeztetéseket ki/be
  - egy kategória alapján válogathatunk, hogy milyen figyelmeztetéseket szeretnénk
  - megadható hogy egy warning FATAL legyen
  - megadható hogy egy warning NONFATAL legyen
    - hasznos lehet, ha mindent FATAL-ra állítottunk
  - modulok szerzői finoman szabályozhatják, hogy milyen figyelmeztetés mikor jeleznek

57

## Operátorok: mintaillesztés

- =~ mintaillesztésre
  - \$variable =~ /pattern/;
  - skalár környezetben igazat ad, ha illeszkedik, hamisat, ha nem
  - my \$string = 'alma';  
print "MATCH" if \$string =~ /ma/;
- != mintaillesztésre
  - \$variable != /pattern/;
  - not (\$variable =~ /pattern/);
- =~ cserére is
  - my \$string = 'hátrány';  
\$string =~ s/h/k/; # kátrány
  - balérték kell
  - visszatérési érték a cserék száma

58

## Operátorok: transliteration

- =~ és tr: transliteration
  - tr/a-z/A-Z/
  - 'alma' =~ tr/a-z/A-Z/;
    - hibás, mert az értéket nem tudja módosítani
    - mindenképpen balérték (lvalue) kell
  - my \$str = 'alma';  
\$str =~ tr/a-z/A-Z/;
  - visszatérési értéke a cserélt karakterek száma
- y egy szinoníma:
  - my \$str = 'alma';  
\$str =~ y/a-z/A-Z/;

59

## Operátorok: qx

- qx//, ``
  - shell parancs végrehajtása
  - a parancs STDOUT -ja a visszatérési érték
    - print "Brightness: " . `cat /proc/acpi/asus/brn`;
    - print "Date now: " . qx/date/;
  - skalár környezetben egy változóban az egész
  - lista környezetben soronként
    - befolyásolható, hogy mit tekintünk sornak
    - \$/, \$INPUT\_RECORD\_SEPARATOR in English
    - ez nem specifikus erre az operátorra!
  - a parancs visszatérési értéke a \$? változóban érhető el
    - \$CHILD\_ERROR in English

60

## Külső programok

- operációs rendszernek futtatásra átadott parancs
- system
  - a paraméterül adott parancsot átadja az operációs rendszernek futtatásra
  - a Perl program futtatása felfüggesztődik addig, amíg a külső program fut
  - ha a program STDOUT-jára nincs szükségem
  - visszatérési érték
    - a program visszatérési értéke
    - -1, ha a programot nem sikerült elindítani
      - a hiba oka a \$! változóban van
    - system('echo', '1', '>/proc/acpi/asus/mled')  
and die "Error running command: \$!\n";
- exec
  - lefuttatja a programot, de soha nem tér vissza

61

## Operátorok: x

- ismétlő operátor
  - repetition operator
- 'a' x 4
  - 'aaaa'
- listákon is működik
  - (1, 2) x 3
    - (1, 2, 1, 2, 1, 2)

62

## File teszt operátorok

- Egy file különböző tulajdonságainak eldöntése
  - egyszerű és tömör
  - shell szerű
- -X\$filename
  - -r: file olvasható
  - -w: file írható
  - -x: file futtatható
  - -e: file létezik
  - -f: egyszerű file
  - -d: könyvtár
  - -l: symbolic link
  - -S: socket
  - stb.

63

## Számok kezelése

- általában a számokat valósként kezeli
- integer pragma jelezheti, hogy használhat egészeket
  - use integer;
  - no integer;
  - ettől még például sqrt(2) értéke valós lesz
- tetszőlegesen nagy egész illetve valós számokat a Math::BigInt illetve a Math::BigFloat modulokkal kezelhetünk
  - van operátor túlterhelés
  - lassúak

64

## perlvar

- perldoc perlvar
- A Perl számos belső globális változót használ
- ezeknek az értékét módosíthatjuk
- pl.
  - output field separator
    - \$,
    - print (1..10);
  - output record separator
    - \$\
    - print 'line';
  - Process ID
    - \$\$
    - print \$\$;

65

## English

- a nehezen megjegyezhető és olvasható változók helyett értelmes név
- gyakran kettő is, egy rövidebb és egy hosszabb
- pl:
  - output field separator: \$,
    - \$OFS
    - \$OUTPUT\_FIELD\_SEPARATOR
  - output record separator: \$\
  - \$ORS
  - \$OUTPUT\_RECORD\_SEPARATOR
  - process id: \$\$
    - \$PID
    - \$PROCESS\_ID
- dokumentációkban gyakran: '\$... in English'

66

## \$\_

- \$ARG in English
- default input és mintaillesztés változó
  - while (<STDIN>) {  
  print "[input] \$\_";  
}
  - \$\_ = 'alma';  
  print "MATCH" if /ma/;
- néhány függvény alapértelmezett paramétere, ha nem adunk meg mászt
  - \$\_ = "output\n";  
  print;
  - \$\_ = 'a,b,c,d,e,f';  
  my @words = split ',';

67

## \$\_

- default iterátor változó foreach ciklusoknál
  - foreach my \$var (@list) {  
  # \$var tartalmazza az aktuális listaelemet  
}
  - foreach (@list) {  
  # \$\_ tartalmazza az aktuális listaelemet  
}
- print \$\_\*\*2, "\n" for (1..5);
  - 1
  - 4
  - 9
  - 16
  - 25

68

## Listák kezelése

- join, map, grep és split együtt rugalmasan kezelik a listákat
  - split: egy sztringet egy elválasztó alapján szétvág, és egy listát állít elő
  - join: egy lista összerakása egy sztringbe
  - map: lista feldolgozása
  - grep: lista elemeinek szűrése
- hiányzik:
  - tartalmazásvizsgálat
  - két listán egyszerre való iterálás lehetősége
  - lista "összegzése", azaz valamely művelet elvégzése a lista minden elemén aggregálva

69

split.pl

## Listák kezelése

- split: az elhatároló egy regexp
  - split(',', \$str);
  - split(/s\*,s\*/, \$str);
  - visszatérési érték az eredmény lista
  - split(',', \$str, 3);
    - korlátozható, hogy legfeljebb hány darabba vágjon szét
  - split(',') a \$\_ változót vágja szét

70

## Listák kezelése

- join: egy lista elemeinek összerakása
  - regexp nem lehet
  - join(',', @list);
  - visszatérési érték a listaelemek összerakása
  - print join(',', 1..10), "\n";
    - 1,2,3,4,5,6,7,8,9,10

71

## Listák kezelése: grep

- grep: listaelemek szűrése
- bemenete egy lista
- kimenete egy másik lista
  - általában nem módosítja a bemeneti listát
- grep EXP, LIST;
  - szűrés menete
    - egyenként vesz minden elemet a listából
    - az aktuális listaelem belekerül a \$\_ változóba
    - kiértékeli rá a megadott kifejezést
    - ha igaz, akkor az elem belekerül az eredmény listába
  - a kifejezésben szerepelhet a \$\_ változó
- a kifejezés lehet például egy regexp is
- ekkor mintaillesztés történik a \$\_ változó értékét felhasználva

72

## Listák kezelése: grep

- `my @filtered = grep /pattern/, @list;`
  - csak azok az elemek, amik illeszkednek
- `my @filtered = grep !/pattern/, @list;`
  - csak azok az elemek, amik NEM illeszkednek
- Vagy blokk, ha több utasítást szeretnénk:
- `grep BLOCK LIST;`
  - nincs vessző!
- `my @filtered = grep { /pattern/ } @list;`
  - ekvivalens az előzővel
  - a blokkban kiértékelt utolsó kifejezés értéke dönt
  - return nem megengedett

## Listák kezelése: map

- `map`: listaelemek feldolgozása
- bemenete egy lista
- kimenete egy másik lista
  - általában nem módosítja a bemeneti listát
- `map EXPR, LIST;`
  - feldolgozás menete:
    - egyenként végigmegy a listaelemeken
    - az aktuális listaelem belekerül a `$_` változóba
    - kiértékeli a kifejezést
    - a kifejezés eredménye kerül az eredmény listába
  - az eredmény lista elemszáma eltérhet a bemeneti lista elemszámától

## Listák kezelése: map

- `my @result = map "[mapped] $_", @list;`
- `my @result = map /^dup/ ? ($_, $_) : $_, @list;`
  - megdupláz minden elemet, ami duppal kezdődik
- `my @filtered = map /pattern/ ? $_ : (), @list;`
  - csökkenti a kimeneti lista elemszámát
  - ugyanaz, mint:
    - `my @filtered = grep /pattern/, @list;`
- `map BLOCK LIST;`
  - nincs vessző!
- `my @result = map {`
  - ...
  - `} @list;`

## Listák kezelése: kombinációk

- `split`, `map`, `grep`, `join` hívásokat kombinálhatjuk
  - pl. `grep` által megszürt listát dolgozunk fel `map`-vel
  - `split->map->join`
- az eredmény mehet egy hash-be is
- `%function_cache =`
  - `map $_ => function($_), @cached_values;`

## Script paraméterek

- A script a parancssori paramétereket az `@ARGV` tömbben kapja
  - nem tartalmazza a file nevét
  - a file neve
    - `$0`
    - `$PROGRAM_NAME` in English
- feldolgozhatjuk kézzel is
- a paraméterek egyszerűbb feldolgozására a `Getopt::Std` és `Getopt::Long` modulok használhatóak
  - az opciókat kiveszi az `@ARGV` tömbből
  - az egyéb paraméterek maradnak

## Több file feldolgozása

- Paraméterként megadott filenevek feldolgozása egyenként
- Fel lehet dolgozni az `@ARGV` tömböt
  - '-' paraméter a standard bemenetet jelenti, ehhez a konvencióhoz érdemes alkalmazkodni
  - ekkor `STDIN`-ről is kell olvasni
- mindezt a funkcionalitást könnyen megvalósíthatjuk az `ARGV` filehandle segítségével
  - paraméterként megadott filenevek feldolgozása
  - `STDIN` beolvasása '-' paraméter esetén

## Több file feldolgozása

- while (<>) {  
  ...  
}
- while (my \$line = <>) {  
  ...  
}
- \$ARGV tartalmazza az éppen megnyitott file nevét
- ha nincs megadva file, az STDIN-ről olvas

## perl futtatása

- perl -e 'print crypt("mypass", "sa"), "\n";'  
  - a megadott perl kód kiértékelése
- perl -ne 'print \$\_\*\$\_\*3.14159, "\n";'  
  - a kód köré:  
  • while (<>) {  
    ...  
  }  
  - a megadott perl kód kiértékelése
- perl -pe \$\_ = "[ARGV] \$\_";  
  - a kód köré:  
  • while (<>) {  
    ...  
  }  
  continue { print or die "-p destination: \$\_\n"; }

## Szignálok kezelése

- %SIG hash használható
- kulcs a szignál neve
- érték
  - 'DEFAULT'
  - 'IGNORE'
  - referencia egy eljárásra
- \$SIG{'INT'} = 'DEFAULT';  
  - eredeti handler visszaállítása
- \$SIG{'INT'} = 'IGNORE';  
  - szignál figyelmen kívül hagyása
- \$SIG{'INT'} = \&int\_handler;
- \$SIG{'INT'} = sub {  
  ...  
};

## \_\_DIE\_\_

- \$SIG{'\_\_DIE\_\_'} = \&handler;
- handler meghívódik minden kivétel fellépése *előtt*
- hívható benne die
- a hiba nem vész el, legfeljebb átalakul
  - csak kezelni tudom a hibát, nem eltüntetni
- eval blokkon belül is meghívódik

## \_\_WARN\_\_

- \$SIG{'\_\_WARN\_\_'} = \&handler;
- handler meghívódik minden futási idejű warning *helyett*
- hívható benne warn
- a warning nem kerül kiírásra, ha nem akarom
- a warning teljes mértékben kezelhető

## Reguláris kifejezések

## Lehetőségek

- mintaillesztés (pattern matching)
  - =~ operátorral
  - illeszkedés eldöntése
  - csoportosított részszttringek elérése
- csere (substitution)
- split
- regexp tárolása egy változóban
- ami hiányzik
  - meg lehessen különböztetni a szó elejét és végét
  - a csoportokra névvel lehessen hivatkozni

85

## Reguláris kifejezések

- / jelek között adjuk meg
  - emiatt a reguláris kifejezésben a / jelet escape-elni kell
  - lehetőség van más elhatároló (delimiter) használatára is
- a mintaillesztés operátora
  - <sztring> =~ /minta/
    - visszatérési értéke skalár környezetben:
      - igaz, ha illeszkedik
      - hamis, ha nem illeszkedik
    - logikai kifejezésekben használható illeszkedés eldöntésére
- a nem illeszkedés operátora
  - <sztring> !~ /minta/

86

## Mintaillesztések

- /world/
  - "Hello world!" =~ /world/
  - print "Match\n" if "Hello world!" =~ /world/;
  - jobban látszik, hogy mi a lényeg, ha:
    - "Hello world!" =~ /world/ and print "Match\n";
    - ez csupán Perl szintaxis kérdése a mintaillesztés szempontjából nincs jelentősége
  - "Hello world!" =~ /World/ and print "Match\n";
    - nem illeszkedik, mert betűérzékeny (case sensitive)
  - "Hello world!" !~ /World/ and print "No match\n";
    - mivel nem illeszkedik, ezért a !~ operátor igazat ad
- /o/
  - "Hello world!" =~ /o/

87

## Mintaillesztések

- "That hat is red" =~ /hat/
  - illeszkedik, de a 'That' szóban található 'hat' részszttringre
- "That hat is red" =~ /\bhat\b/
  - ez már a 'hat' szóra illeszkedik

88

## Mintaillesztések

- \$ a sor végét jelenti, többféleképpen is viselkedhet
  - "Simple line" =~ /line\$/ and print "MATCH";
    - \$ illeszkedik a sztring végére
  - "Simple line\n" =~ /line\$/ and print "MATCH";
    - \$ illeszkedik az újsor karakterre és a sztring végére
  - "Simple line" =~ /line\n\$/ or print "NO MATCH";
    - a sztringben nem szerepel újsor karakter, ezért nem illeszkedik
  - "Simple line\n" =~ /line\n\$/ and print "MATCH";
    - illeszkedik az újsor karakterre és a sztring végére

89

## Elhatároló megváltoztatása

- ha az elhatároló (delimiter) szerepel
  - \$shebang = '#!/usr/bin/perl -w';
  - \$shebang =~ /\#/usr/bin/perl/ and print "Perl";
- elhatároló (delimiter) megváltoztatása
  - jelölni kell, hogy mintaillesztést szeretnénk
  - pattern matching: m
    - m utáni karakter jelzi a minta (pattern) végét
  - \$shebang =~ m#/usr/bin/perl# and print "Perl";
    - sokféle karakter lehet elválasztó
      - m{/usr/bin/perl}
      - m[/usr/bin/perl}
      - m(/usr/bin/perl)
      - m[|usr/bin/perl}

90

## Változóhelyettesítés

- az a feature, amivel a Perl sztringekbe a változók értékét helyettesíti
  - \$who = 'Peti';  
print "Hello \$who";
    - Hello Peti
- minták esetében is működik
  - "Hello Petike" =~ /\$who/ and print "MATCH";
    - illeszkedik a 'Petike'-ben lévő 'Peti'-re
  - "Hello \$who" =~ /\$who/ and print "MATCH";
  - 'Hello \$who' =~ /\\$who/ and print "MATCH";
    - a sztringben az aposztrófok miatt, a mintában a \ miatt nincs helyettesítés => ezért illeszkednek
- elkerülhető, ha aposztróf a határoló (delimiter)

91

## Metakarakterek

- {} [] ()
- ^ \$ | \* + ? \
- ha illeszteni szeretnénk ezekre a karakterekre, akkor escape-elni kell a \ karakterrel
- "2+2" =~ /2+2/ or print "NO MATCH";
- "2+2" =~ /2\+2/ and print "MATCH";
- "C:\windows" =~ /C:\\windows/ and print "MATCH";

92

## Regex példák

example.pl

- eldöntendő kérdés a felhasználónak
- \$answer tartalmazza a választ
- /y|n/
  - print "Please enter y or n"  
unless \$answer =~ /y|n/;
  - \$answer =~ /y/ ? agreed() : exit(0);
  - \$answer = 'no way'
    - ekkor is illeszkedne
- /^y|n\$/
  - az előző problémát megoldja
- /^(y|n)/
  - rugalmasabb
  - megengedett a yes, no, y, n, yeah, no way
  - de az is, hogy yellow, vard, noon, nun

93

## Karakterosztályok

- [] karakterek között felsorolva
- /[cr]at/
  - cat
  - rat
- a karakterek sorrendje nem befolyásolja az illesztés sorrendjét
  - "abc" =~ /[cab]/ and print "MATCH";
    - az a karakterre illeszkedik, nem pedig a c-re!
    - azaz: a sztring első karakterére és nem a harmadikra
- /[yY][eE][sS]/
  - yes, Yes, YES, stb.
- - segít intervallumok (range) reprezentálásában
  - '12' =~ /^[0-9]/ and print "MATCH";
  - '-12' =~ /^[-0-9]/ and print "MATCH";

94

## Karakterosztályok

- ^ egy karakterosztályon belül nem a sztring elejét jelöli
- a felsorolt karakterek komplementere illeszkedhet
- 'alma' =~ /^[^abc]/ or print "NO MATCH";
  - az első kalap jelöli a sztring elejét
  - a második a karakterhalmaz negálását
    - minden illeszkedik, ami nem 'a' vagy 'b' vagy 'c'
- ha szeretnénk egy kalapra (caret) illeszteni:
  - a [] jelek között ne a kalap legyen az első
    - ha ez lenne az egyetlen, akkor /\^/
      - 'kalap:' =~ /[^\^]/ and print "MATCH";
      - 'kalap:' =~ /[\^]/ and print "MATCH";
      - 'kalap:\*' =~ /[^\^]/ or print "NO MATCH";

95

## Előre definiált karakterosztályok

- [0-9]
  - számjegyek
- [a-zA-Z0-9\_]
  - alfanumerikus és aláhúzás (underscore)
  - szókérekek
- hasonló kifejezések helyett előre definiált karakterosztályok
  - pl. \w, \d
- használhatók egy karakterosztályon belül is
  - hexadecimális számjegyek osztálya
    - \da-fA-F
- csak rövidítések!
  - ugyanúgy viselkednek, mintha [] között megadtuk volna a lehetséges karaktereket

96



## Előre definiált karakterosztályok

- \d számjegy: [0-9]
- \D nem számjegy: [^0-9]
- \w szókéreker: [a-zA-Z0-9\_]
- \W nem szókéreker: [^a-zA-Z0-9\_]
- \s whitespace: [ \t\n\r\f]
- \S nem whitespace: [^ \t\n\r\f]

97

## POSIX karakterosztályok

- POSIX karakterosztályok is használhatók
- a következő osztályok használhatók
  - alnum, alpha, ascii, blank, cntrl, digit, graph, lower, print, punct, space, upper, word, xdigit
- így például az előző példa helyett
  - [\da-f-F]
  - [[:xdigit:]]
- keveredhetnek is
  - szabadon tördelhető hexadecimális érték
    - /^[[:xdigit:]\s]\*\$/

98

## Ez vagy az: alternatívák

- alternation
- \$traffice\_light =~ /red|yellow/ and brake();
- az alternatívák sorrendje befolyásolhatja azt, hogy mire illeszt a Perl
  - akkor nem befolyásolja, ha egyik alternatíva sem prefixe a másiknak
- 'cats and dogs' =~ /cat|dog/ and print "MATCH";
  - 'cat'-re illeszkedik
- 'cats and dogs' =~ /dog|cat/ and print "MATCH";
  - szintén 'cat'-re illeszkedik
- 'cats' =~ /c|ca|cat|cats/ and print "MATCH";
  - illeszkedik ugyan
  - de nincs értelme az alternatíváknak, ha illeszkedik. akkor mindig az elsőre

99

## Ez vagy az: alternatívák

- 'cats' =~ /cats|cat|ca|c/ and print "MATCH";
  - illeszkedik
  - 'cats' illeszkedik, mert ez a felsorolásban az első
- az egyes alternatívákban a ^ és a \$ külön szerepeltethető
  - /^almaf\$/
- üres alternatíva is megengedett
  - /Impala|/
    - vagy Impala, vagy semmi
    - 'Impala' =~ /Impala|/ and print "MATCH";
    - "" =~ /Impala|/ and print "MATCH";
    - 'Suzuki' =~ /Impala|/ and print "MATCH";
  - /^Impala\$|^\$/
    - "" =~ /^Impala\$|^\$/ and print "MATCH";
    - 'Suzuki' =~ /^Impala\$|^\$/ or print "Vidd innen!";

100

## Csoportosítások

- kettős szerepe van
  - a regex egy részének atomi kezelése
    - alternatívákhoz
    - ismétlődésekhez
  - az illeszkedő részsstring felhasználása
- /house(cat|keeper)/
  - 'housecat' =~ /house(cat|keeper)/ and print "MATCH";
  - 'housekeeper' =~ /house(cat|keeper)/ and print "MATCH";
- egymásba ágyazható!
  - /house(cat(s))|/
    - house
    - housecat
    - housecats

101

## Csoportok elérése

- /Festés: \d+ Ft/
  - nem elég, hogy tudom, hogy illeszkedik
  - tudni szeretném, hogy hogyan
- /Festés: (\d+) Ft/
  - a csoportosított regexp részre illeszkedő részsstringek elérhetőek később is
- 'Festés: 11000 Ft' =~ /Festés: (\d+) Ft/
  - a 11000, amit szeretnék elérni
  - a mintaillesztés után a \$1 változóban érhető el
  - ha több csoport van, akkor \$1, \$2, \$3, ...

re-extract.pl

102

## Beágyazott csoportok elérése

- date
  - Sun Feb 22 18:13:09 CET 2004
- szeretném az időt egyben is és darabokban is megkapni
- my (\$time, \$hour, \$minute, \$second) =  
\$date =~ /((\d\d):(\d\d):(\d\d))/;  
#           12       3       4
  - a csoportok pozícióját a nyitó zárójel határozza meg

103

## Visszahivatkozások egy mintán belül

- backreference
- egy csoport ismételt előfordulására szeretnék illeszteni
- a \$1, \$2, \$3, ... változók csak a mintán kívül használhatóak
- \1, \2, \3, ... csak egy mintán belül használhatóak
- pl. dátumintervallum megadásánál szeretném, ha mindkét dátum ugyanabban az évben lenne
  - dátum formátuma: YYYY.MM.DD
  - \$interval =~ /(\d{4})\.\d\d.\d\d. - \1.\d\d.\d\d./;
  - '2004.01.12. - 2004.03.12.' illeszkedik
  - '2003.12.22. - 2004.03.12.' nem illeszkedik
  - mellékhatásként a \$1 értéke az adott év lesz

104

## Ismétlődések

- a megelőző atom előfordulásának száma írható le vele
- a{n,m}: legalább n, legfeljebb m
- a{n,}: legalább n
- a{n}: pontosan n
- a?: 0 vagy 1
  - a{0,1}
- a\*: 0 vagy több
  - a{0,}
- a+: 1 vagy több
  - a{1,}

105

## Ismétlődések

re-extract.pl

- /\d{2,4}/
  - évszámok: de sajnos a 3 jegyű is
- /\d{2}|\d{4}/
  - ez nem jó, mindig az első alternatíva választódik ki
- /\d{4}|\d{2}/
  - kettő vagy négyjegyű
- csoportosítással az ismételhető megelőző atom fogalma sokkal rugalmasabbá válik
- emailcímek
  - /\w+@(\w+\.)+\w+/
    - john@elte.hu
    - john@inf.elte.hu
    - john.doe@inf.elte.hu nem jó!

106

## Ismétlődések

- /(\w+\.)\*\w+@(\w+\.)+\w+/
  - john.doe@inf.elte.hu
  - hannibal.lecter@i-eat-you.com nem jó
- /(\w+\.)\*\w+@(\w[\w-]+\w\.)+\w+/
  - hannibal.lecter@i-eat-you.com
- /(\w+\.)\*\w+@(\w[\w-]+\w\.)+\w{2,7}/
  - ha még tudjuk, hogy a fő domain hossza 2 és 7 karakter közé esik
  - mellékhatásként \$1, \$2 módosulhat
  - (?...): csoportosítás capturing nélkül
    - /(?:\w+\.)\*\w+@(\w[\w-]+\w\.)+\w{2,7}/
- ezek a megoldások **nagyon** hiányosak
  - csak illusztrációnak felelnek meg
- teljes megoldások elérhetőek az Interneten

107

## Greediness - mohóság

- az ismétlők mohók: olyan sokat 'markolnak fel', amennyit bírnak
- /(\d\*)(\d\*)/
  - '123456' =~ /(\d\*)(\d\*)/;
  - \$1: '123456'
  - \$2: undef
  - a második mindig undef lesz
- például szeretnék egy file abszolút nevéből levágni a könyvtárat
- \$path =~ m#(.\*)([^\/\*]\*)#;
- az első csoportban a \* mohó (greedy), ezért egészen az utolsó / jelig mindent illeszt

108

## Greediness - mohóság

- `my ($dir, $file) = '/usr/local/bin/perl' =~ m#(.*)([/\]*)#;`
  - `$dir: '/usr/local/bin'`
  - `$file: 'perl'`
- `m#(.*)(.)*#;`
  - mindkettő mohó, de amelyik előbb szerepel, az van előnyben
  - a második csoport mohósága háttérbe szorul
  - ez a kifejezés ugyanazt eredményezi, mint az előző

109

## Greediness - mohóság

- `m#^(.*)/([^\/]*)$#;`
  - `'/usr/local/bin/'` nem illeszkedik
- `m#^(.*)/(.+$)#;`
  - `'/usr/local/bin/'` illeszkedik
- `my ($dir, $file) = '/usr/local/bin/' =~ m#^(.*)/(.+$)#;`
  - `$dir: '/usr/local'`
  - `$file: 'bin/'`
  - az első csoport illeszkedésének hosszát az utána következők mohósága nem, de az igénye befolyásolja

110

## Greediness - mohóság

- cél: az első komponens a path-ban
- `m#(.*?)#;`
  - nem jó, mert az ismétlődő \* mohó (greedy)
  - `my ($dir) = '/usr/local/bin/perl' =~ m#(.*?)#;`
    - `$dir: '/usr/local/bin'`
- első megoldás:
  - `m#(?:[/\]*)#;`
- másik megoldás: fogjuk vissza a mohóságot
  - ? az ismétlődő után visszafogja a mohóságot, az ismétlés nem lesz mohó (non-greedy)
  - `m#(.*?)#;`
  - `my ($dir) = '/usr/local/bin/perl' =~ m#(.*?)#;`
    - `$dir: undef`
    - ez nagyon kevésbé lesz mohó: 0 hosszan illeszkedik

111

## Greediness - mohóság

- `m#(.*?)#;`
  - `my ($dir) = '/usr/local/bin/perl' =~ m#(.*?)#;`
    - `$dir: '/usr'`
- `'aaaa' =~ /(a{2,4})/;`
  - `$1: 'aaaa'`
- `'aaa' =~ /(a{2,4})/;`
  - `$1: 'aaa'`
- `'aaaa' =~ /(a{2,4}?)/;`
  - `$1: 'aa'`

112

## Reguláris kifejezés változóban

- Egy reguláris kifejezést eltárolhatunk egy változóban is
  - felhasználható később
  - átadható paraméterként
  - stb.
- a `qr//` operátor segít
- `$regex = qr/(w+): (\d+) cm/;`
- `'Pete: 178 cm' =~ /$regex/;`
  - `$1: 'Pete'`
  - `$2: '178'`

113

## Módosítók

- Módosítókkal finomíthatjuk a mintaillesztés módját
  - a mintát lezáró elhatároló után kell
- `i`: nem betűérzékeny (case insensitive) illesztés
  - `$answer =~ /yes/i`
    - `yes, Yes, YES, stb.`

114

## Módosítók

- **m:** multiline
  - ^ illeszkedik egy többsoros sztringben a sor elejére
  - \$ illeszkedik egy többsoros sztringben a sor végére
  - "First line\n99" =~ /^d+/  
or print "NO MATCH";
  - "First line\n99" =~ /^d+/m  
and print "MATCH";
  - \A mindig a sztring elejére illeszkedik
  - \Z mindig a sztring végére illeszkedik

115

## Módosítók

- **s:** single line
  - . alapértelmezésben nem illeszkedik az újsor karakterre (\n)
  - az s módosítóval viszont igen
  - "Pete:\n17" =~ /\w+\d+/  
or print "NO MATCH";
  - "Pete:\n17" =~ /\w+\d+/s  
and print "MATCH";
  - ^ itt a sztring elejére illeszkedik

116

## Módosítók

- **x:** extended legibility (olvashatóság)
  - minden fehér szóközt figyelmen kívül hagy, ami nincs escape-elve
  - megjegyzések is elhelyezhetőek
- /
  - (?:\w+\.)\* # opcionális rész név
  - \w+ # a név része, vagy azonosító
  - @ # @ jel
  - (?:\w
  - [\w-]+ # utána szó karakter, vagy -
  - \w\. # . a domain-ek között
  - )+ # host és domain nevek
  - \w{2,7} # fő domain

/x

117

re-global.pl

## Módosítók

- **g:** globális illesztés
  - minden illesztést megkeres
  - skalár ill. lista környezetben eltérően viselkedik
- /(\d+)/g
  - skalár környezetben
    - minden egyes illesztés egy újabb illesztést ad vissza
    - a \$1, \$2, ... változókat kell használni
    - reset:
      - pos(\$string) = 0;
      - () = \$string =~ /(\d+)/g;
  - lista környezetben
    - visszaadja az összes illesztést

118

## Kapcsolódó változók

- \$\_ (\$ARG in English)
- ezek használata minden regex műveletet lelassít az egész programban
  - \$& (\$MATCH in English)
  - \$` (\$PREMATCH in English)
  - \$' (\$POSTMATCH in English)
- \$+ (\$LAST\_PAREN\_MATCH in English)
  - /Version: (.\*)|Revision: (.\*)/ && (\$rev = \$+);
  - az eredmény \$1-ben és \$2-ben is lehet, attól függően, hogy melyik alternatíva illeszkedett

119

## Kapcsolódó változók

- @- (@LAST\_MATCH\_START in English)
  - az illesztett csoportok kezdetének offszetje
  - \$-[0] az egész illesztés kezdetének offszetje
  - captured csoportokra vonatkozik
    - azaz: (?:...) csoportok itt nem szerepelnek
- @+ (@LAST\_MATCH\_END in English)
  - az illesztett csoportok végeinek offszetje
  - \$+[0] az egész illesztés végének offszetje
  - captured csoportokra vonatkozik
    - azaz: (?:...) csoportok itt nem szerepelnek
- 'john.doe@inf.elte.hu' =~  
/(\w+\.)\*\w+@(\w[\w-]+\w\.)+\w{2,7}/;
  - @-: (0, 0, 13)
  - @+: (20, 5, 18)

120

## Cserék

- `=~ s///` operátorokkal
- `s:` substitution
- `$string =~ s/mit/mire/;`
  - `$string` megváltozik
  - visszatérési érték a cserék száma
  - *mit* egy regex
  - *mire* nem regex, csak egy double quoted string
- `$str = '2 + 2 = 4';`
  - `$str =~ s/2/two/;`
    - `$str:` 'two + 2 = 4';
  - `$str =~ s/2/two/g;`
    - `$str:` 'two + two = 4';
    - `g:` globális csere, minden előfordulást cserél

121

## Cserék

- két szó felcserélése
- `$string = 'one two';`  
`$string =~ s/(\w+) (\w+)/$2 $1/;`
  - `$string:` 'two one'
- `$string = 'one two three four';`  
`$string =~ s/(\w+) (\w+)/$2 $1/g;`
  - `$string:` 'two one four three'
- mintaillesztésnél használatos módosítóknak itt is ugyanaz a funkciójuk
- `e:` evaluate
  - kiértékeli a cserét, mint kifejezést, és annak az értékével helyettesít
  - ezzel módosítható az illesztett minta

122