

## Modulok, csomagok

1

## Csomagok

- package, modul, csomag
- névtérzennyezés (namespace pollution) helyett külön névterek használandóak
- minden csomagnak külön névtere van
  - külön szimbólumtábla
  - azonosítókat foglal magába
- csomag elválasztó (package delimiter)
  - ::
  - régen: '
- csomagbeli változók scope-ja a csomagra vonatkozik
  - kivéve: Perl belső változói
- a főprogram mindig a main csomagban van
- nincs globális változó

2

## Csomagok

- kívülről hivatkozhatók tetszőleges csomag tetszőleges azonosítójára
  - MyPackage nevű csomagban lévő \$scalar változóra hivatkozás:
  - \$MyPackage::scalar
- nincs korlátozás arra, hogy mit érhetnek el, és mit nem
- egy másik csomagból hivatkozhatók a főprogram változóira
  - a main package-ben vannak
  - \$main::scalar\_in\_main
  - \$::scalar\_in\_main
- \_\_PACKAGE\_\_ mindig az aktuális csomag neve

3

## Csomagok használata

- egy csomag exportálhat bizonyos azonosítókat
  - belekerülnek az őt használó csomag névtérébe
  - import
- use MyPackage;
  - fordítási idejű
  - importál
- require MyPackage;
  - futási idejű
  - nem importál
    - import MyPackage;
- ezután használhatom az adott csomagot
  - a csomag névén keresztül a package delimiter segítségével
  - az exportált azonosítókat közvetlenül is

4

## our

- csomagszintű változók definiálására használható
- az adott lexikális blokk végéig látható
  - adott esetben akár csomaghatárok között is
- hasznos, ha use strict 'vars'; miatt deklarálni kell a változókat, de my nem megfelelő
- package MyPackage;  
our \$foo = 10;  
...

5

## Csomagok definiálása

- egy csomag definíciója a package kulcsszóval történik
  - package MyPackage;
- a csomag végét több dolog jelentheti
  - következő csomag kezdete
    - ezzel több csomagot helyezhetek el egyetlen fájlban
  - blokk vége
    - egy blokkon belül kezdődhet több csomag
  - az eval sztring vége
    - egy dinamikusan futtatott kód nem hat ki az utána következő sorokra
  - file vége
- a csomagok fordítási időben jönnek létre és az élettartamuk nem kötődik a definíciós helyéhez
  - kivétel: eval sztringek

6

## Csomagok definiálása

- gyakorlati használat
  - egy file egy csomag
    - ha egy file-ban több csomag van, akkor a file nevével nem megegyezőket a file nevével megegyező privát célokra használja
  - file neve: csomag.pm
    - pm: Perl Module
  - hierarchikusan rendezhetők a csomagok
    - a könyvtárhierarchia felhasználható
    - use My::Package;
      - My/Package.pm
- a értelmező lefuttatja a csomagban levő kódot
  - inicializációra BEGIN blokkot célszerű használni

7

## Csomagok definiálása

- require igaz értéket vár eredményül
  - egy csomagban lefuttatott kódnak igaz értéket kell visszaadnia
  - általában ezt a sort használjuk:  
1;
- csomagok nevét nagybetűvel kezdjük, minden szó első karatere nagybetű
  - MyFirstPackage

8

MyPackage.pm  
call-mypackage.pl

## Csomagok tulajdonságai

- -w kapcsoló az egész programra vonatkozik
  - minden használt csomagban warning-ok lépnek fel nem megfelelő kód esetén
- pragmak nem az egész programra vonatkoznak
  - a pragmatól függően vonatkozhatnak egy file-ra, vagy egy csomagra
  - pl. a főprogramban használt strict pragma nem vonatkozik a használt modulokra

9

## Csomag konstruktorok

- BEGIN blokkban definiálhatjuk ezeket az utasításokat
  - BEGIN {  
...  
}
- még az interpreter futásának megkezdése előtt lefutnak
  - ASAP
  - még azelőtt lefut, hogy a file többi része feldolgozásra kerülne
- FIFO sorrendben
  - az előbb definiált BEGIN blokk előbb fut le

10

## Csomag konstruktorok

- CHECK blokkok
  - CHECK {  
...  
}
  - LIFO sorrendben futnak le
  - rögtön a fordítási fázis befejezése után, de még a futtatás megkezdése előtt futnak le
- INIT blokkok
  - INIT {  
...  
}
  - FIFO sorrendben futnak le
  - pontosan a futtatás megkezdése előtt futnak le
  - a futtatás első lépése

11

## Csomag konstruktorok

- CHECK és INIT blokkok használhatóak arra, hogy a fordítás és a futtatás közötti állapotot elkapjuk
- a CHECK blokkok mindig lefutnak az INIT blokkok előtt
- perl -c
  - csak fordítás, a futtatási fázis nem indul el
    - ha minden rendben, egy <filenev>: Syntax OK üzenetet kapunk
  - a BEGIN és a CHECK blokkok lefutnak
    - minden csomagban

12

## Csomag destruktorkok

- END blokkok
- END {  
...  
}
- végrehajtásuk
  - LIFO: definíciójuk fordított sorrendjében történik
  - ASLP: amilyen későn csak lehet
- die esetén is lefut
- nem fut le, ha
  - exec függvénnyel egy másik programot indítunk
  - kezeletlen szignál miatt lép ki a program

13

## Scriptekben

- Használható egyszerű scriptekben
- a főprogram a main csomagban van
  - használható BEGIN, END, stb. blokk
- -n opció esetén inicializációt végezhetünk a BEGIN blokkban

14

## Export

- a csomagokban levő azonosítók külön névtérben vannak
  - rájuk történő hivatkozás a csomag explicit megjelölésével lehetséges
  - POSIX::strftime(...)
- lehetséges, hogy saját névtérünkbe importáljunk bizonyos azonosítókat
  - azt, hogy mit lehet importálni, a csomag szabja meg
  - azt, hogy mi kerül importálásra, azt a csomag használója szabhatja meg
- Exporter modul
  - require 'Exporter';
  - @ISA = qw(Exporter);

15

## Export

- @EXPORT
  - ebbe a tömbbe helyezzük azokat az azonosítókat, amiket alapértelmezésben szeretnénk exportálni
- @EXPORT = qw(welcome);
  - a welcome eljárást exportálja
- use MyPackage;
  - a welcome importálódik a használó modul névtérébe
- use MyPackage ();
  - semmi nem importálódik a használó modul névtérébe

16

## Export

- @EXPORT\_OK
  - azokat az azonosítókat kell beletenni, amik importálására lehetőséget akarunk biztosítani
- @EXPORT\_OK = qw(goodbye);
  - a goodbye importálható lesz
- use MyPackage;
  - nem importálja a goodbye eljárást
- use MyPackage qw(goodbye);
  - explicit jelzés kell
    - az @EXPORT tömbbeli azonosítók sem importálódnak automatikusan
  - importálja a goodbye eljárást

17

## Export

- %EXPORT\_TAGS
  - csoportosítható vele néhány exportálandó illetve exportálható azonosító
- %EXPORT\_TAGS = (  
  greetings => [ qw(welcome goodbye) ],  
);
- use MyPackage;
  - a fenti hash-nek nincs hatása
  - a működést az @EXPORT és az @EXPORT\_OK tömbök befolyásolják
- use MyPackage qw(:greetings);
  - importálja a welcome és a goodbye eljárásokat is
- use MyPackage qw(:DEFAULT goodbye);

18

## Modulok helye

- hol keresi a Perl a modulokat?
  - aktuális könyvtár
  - perl -V
  - module\_name.pm file-ban
- @INC tömb szabályozza
  - egy path-t kell beletenni
  - push
  - futási időben értékelődik ki
- lib pragma
  - a fentiek szebb megoldása
  - fordítása időben kiértékelődik

19

## Objektum-orientált Perl

20

## Objektum-orientált fogalmak

- osztály
  - egységbe záras (encapsulation)
  - adatelrejtés
- példány (objektum)
- adattag
- metódus
- öröklődés
  - egyszeres
  - többszörös
- polimorfizmus
  - dinamikus összekapcsolás
- konstruktorok
- destruktorok

21

## Osztályok

- egy osztály nem más mint egy csomag (package)
- nincs külön kulcsszó
- speciális függvények a csomagban
  - osztálymetódusok
  - példánymetódusok
- külön namespace
  - egységbe záras
- adatelrejtés nem kérhető
  - nincs public, private, stb. kulcsszó
  - konvenciók
    - \_ jellel kezdődő azonosítók private-nak tekintendők
    - hasonló a helyzet, mint a műszaki cikkeknel :)

22

## Metódushívás

- többféle metódushívási szintaxis létezik
  - method Class(params);
    - a zárójel itt tulajdonképpen opcionális
  - Class->method(params);
- első paraméter az osztály neve

23

## Objektumok

- Az objektum Perl-ben nem más, mint egy speciális referencia
  - blessed
  - az objektum és a típusinformációja együtt kezelődik
- Perl adattípusok
  - skalár
    - referenciák
      - szimbolikus referenciák
      - valódi referenciák (hard references)
        - objektumok
    - számok, sztringek
  - lista, hash
- a referencia tetszőleges típusra mutathat
  - tipikusan hash lesz

24

## Objektumok

- `bless REF, TYPE;`
  - a visszatérési érték egy objektum
  - a megadott referenciára mutat
  - `TYPE` a típusa
  - opcionális zárójelek
- `TYPE`
  - egy csomag neve
  - sztringként megadva
  - osztályszintű metódusoknál ez lesz az első paraméter
- objektumok mezőinek kezelését nem a csomag változóiban kell megvalósítani
- a mezők kezelése az osztály definiálójának dolga
  - eszközt a referenciák adnak

25

## Objektumok

- `my $not_yet_obj = {`
  - `'field1' => 'value1',`
  - `};`
- `my $object = bless $not_yet_obj, 'MyClass';`
  - így történik a példányosítás
  - semmi nem akadályoz meg abban, hogy egyszerre legyenek objektumaim, amik listára és hash-re mutatnak
    - az osztály metódusainak ezt tudni kell kezelni
    - felesleges bonyolítás
  - ezt akár az osztályon kívül is megtehetem
  - a megadott változó megváltozik
    - a visszatérési érték kényelmi funkció csupán

26

oo-objects.pl

## Objektumok

- `bless REF`
  - ha nincs megadva típus, akkor az aktuális csomag lesz az objektum típusa
  - öröklődésnél problémákat okoz
- metódushívási szintaxis
  - `$obj->method(...);`
  - `method $obj;`
    - mást jelent: method függvény meghívása `$obj` paraméterrel
    - nem objektum-orientált módon kezelődik
- a `$obj` lesz a metódus első paramétere
- ezt mindig át kell venni, ha példánymetódust írunk
  - `my ($self) = @_;`

27

## Konstruktorok

- osztály konstruktorai
  - `BEGIN`, `INIT`, `CHECK`
- objektum konstruktorok feladata Perl-ben
  - `bless`
  - az objektum struktúrájának felépítése
  - öröklődés kezelése
  - akár `copy` `ctort` is írhatunk
- Perl-ben nincs `new` kulcsszó
  - nincs automatikus mechanizmus a példányosításra
- `new` metódust szokás erre a célra használni
  - `new Class(...);`
  - `Class->new(...);`

28

lib/Color.pm  
oo-colors.pl

## Konstruktorok

- általában osztálynévre hívjuk meg őket
- a metódus első paramétere a csomag (osztály) neve lesz
  - ez felhasználható, mint típus
  - ha a csomag nevét használnánk fel, az öröklődés során nem lenne megfelelő az objektum típusa
- az objektum mezői
  - egy hash-ben tárolva imitálhatjuk az objektum mezőit
  - a konstruktor feladata ennek a felépítése is

29

lib/Colorv2.pm  
oo-colorsv2.pl

## Polimorf metódusok

- többféleképp viselkedő metódusok könnyedén írhatóak
- getter/setter metódusok helyett egy
  - `get_color()`
  - `set_color()`
- `color()`
  - ha nincs paraméter, akkor getter funkció
  - ha van, akkor setter funkció
  - a paraméterek számának vizsgálata

30

## Általánosabb konstruktorok

- `Class->new()`
  - első paraméter az osztály neve
  - `'Class'`
- `$obj->new()`
  - első paraméter `$obj`
  - referencia
  - `ref($obj)` igaz lesz
    - sőt: az objektum típusát adja meg
- opcionális írhatunk `copy ctor`
  - ha az első paraméter referencia (azaz objektum)
  - az adatszerkezet lemásolása

31

## Destruktorok

- DESTROY metódus
  - `sub DESTROY {`
    - ...
    - `}`
- közvetlenül az objektum felszabadulása előtt hívódik meg
- egyetlen paramétere az objektum

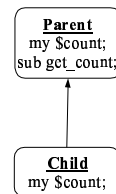
32

## Osztályváltozók

- a csomag változói
  - `my`
  - `our` – kevésbé jó
- minden metódus látja

33

## Osztályváltozók



- probléma az előző implementációval
- a `Child` osztályban, vagy annak egy példányára meghívott `get_count` metódus a `Parent $count` változóját fogja kezelni
  - a metódus a definiálás helyének megfelelő környezetben fut le
- megoldás
  - minden objektum tartalmazzon egy referenciát a globális változóra

34

## Osztály destruktorkok

- `END` blokkok
- `END {`
  - ...
  - `}`
- az osztály megszűnésekor futnak le

35

## Öröklődés

- szintaktikailag nem támogatott
  - nincs kulcsszó az ISA reláció jelölésére
- szemantikailag támogatott
- `@ISA` tömb
  - minden csomagban lehet
  - osztályneveket tartalmaz
  - a `Perl`, ha nem talál egy metódust, akkor megnézi ezt a tömböt
  - ha a benne levő osztályban van ilyen metódus, akkor meghívja
  - ha nincs, akkor tovább keres
  - ha nem sikerül megtalálni a kért metódust, az futási idejű hiba

36

## Öröklődés

- metódushívási szintaxis esetén működik az @ISA tömbben való keresés
- Class->method(...);
  - metódushívási szintaxis
  - metódushívási szemantika
- Class::method(...);
  - függvényhívási szemantika
  - az osztály neve nem adódik át paraméterként!
  - nincs öröklődés

37

lib/ParentClass.pm  
lib/ChildClass.pm  
oo-inheritance.p

## Öröklődés

- use Parent;  
  @ISA = qw(Parent);  
  ...
- az osztály definíciója során meg kell adni
- @ISA nem lehet lexikális (my) változó
- use strict;  
  use Parent;  
  our @ISA = qw(Parent);
- base pragma
  - modern mód a szülők megadására
  - egysoros
  - nem kell az @ISA tömbbel sem bajlódni
- use base qw(Parent);

38

## Öröklődés

- az öröklődés csak metódusok öröklését jelenti
  - adattagokat nem öröklünk automatikusan
  - ehhez szükséges a konstruktorban a megfelelő logika
  - osztályváltozók sem öröklődnek

39

## Override

- override természetesen működik
  - polimorfizmus
- \$obj->SomeClass::method(...);
  - explicit megmondhatjuk, hogy melyik osztályban levő metódust használja
  - pl. a gyerekosztály meghívhatja egy felüldefiniált metódusban a szülő metódusát
    - \$self->MyParent::method(...);
- SUPER pszeudo-osztály
  - \$self->SUPER::method(...);

40

## UNIVERSAL

lib/Parent.pm  
lib/Child.pm  
oo-universal.pl

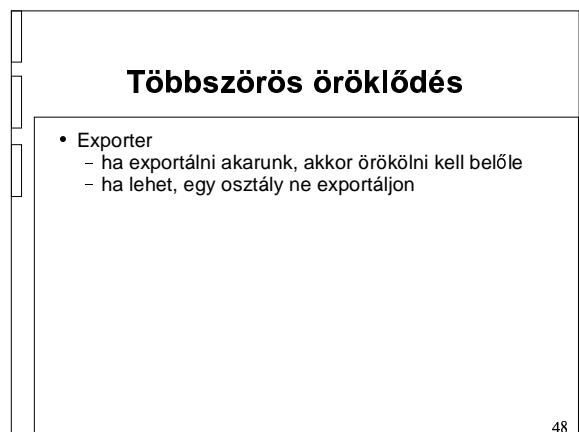
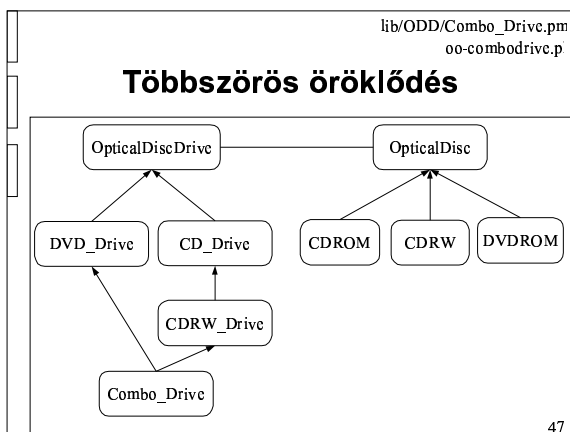
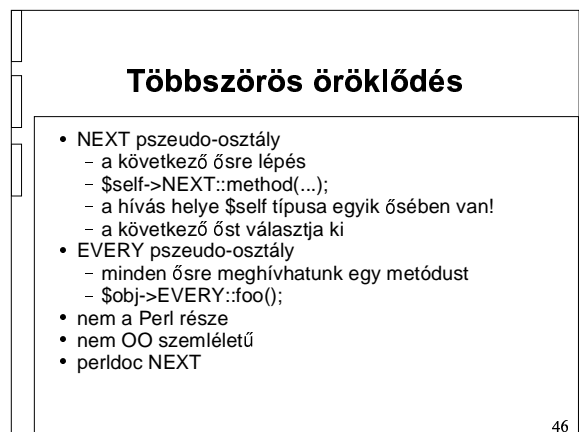
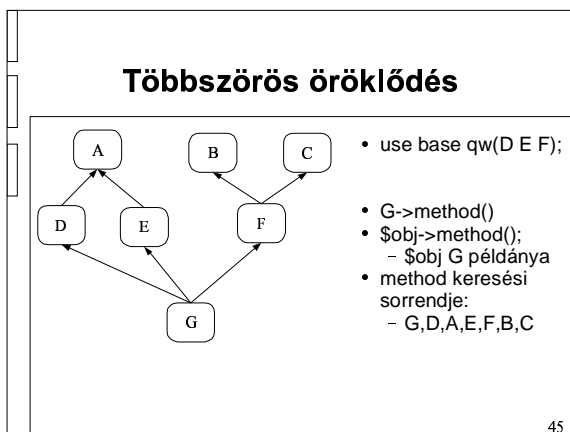
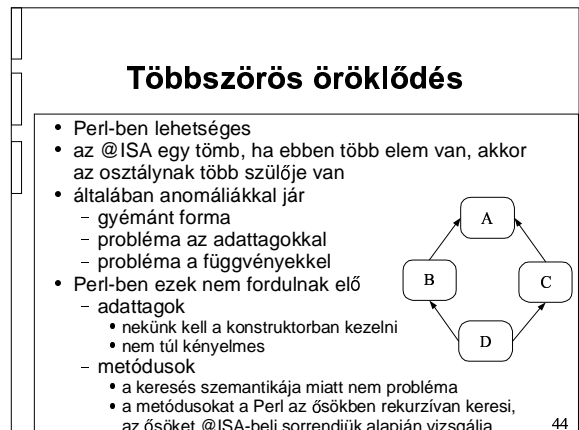
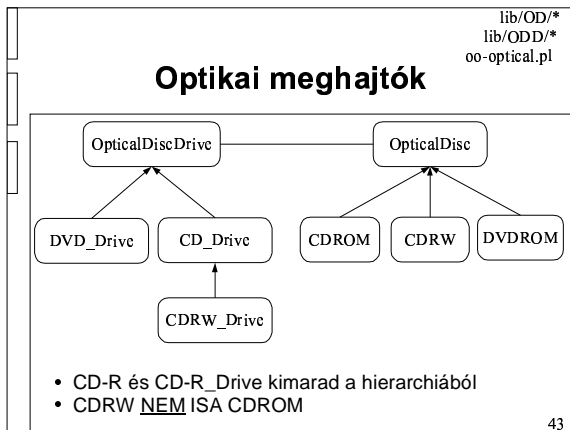
- minden osztály közös őse
- VERSION
  - a modul verziója
  - a modulban definiálni kell a \$VERSION változót
  - \$VERSION nem lehet lexikális
- isa
  - ref(\$obj): megadja az objektum típusát, de semmit nem mond a szülőkről
  - ISA reláció fennállásának eldöntése
- can
  - eldönthető, hogy egy objektumra vagy osztályra meghívható-e a paraméterül adott metódus
    - ha igen, kód referenciát ad vissza!
  - figyelembe veszi az öröklési relációkat

41

## Absztrakt osztályok, metódusok

- nyelvi eszköz nincs rá
- futási idejű hibát azért dobhatunk
  - new konstruktorban a típust kell megvizsgálni
  - ha a típus a definiálás csomagja, akkor hibát dobunk
- absztrakt metódus is imitálható
- az ősoosztályban levő metódus mindenképpen hibát dob
  - ha a hívás olyan osztályra, vagy annak példányára történik, aki nem definiálta felül, akkor hiba lép fel

42





## Class::Struct

- Perl-ben nincs rekord típus
- ha szükségünk van rá, és adatelrejtést is szeretnénk, akkor egy osztályt kell implementálni
  - adattagok kezelése
  - konstruktor
  - egyéb szolgáltatások
- Class::Struct modul generál nekünk ilyen metódusokat
  - egyéb szolgáltatásokat kell csak megírni
- perldoc Class::Struct

49

## Reprezentáció elrejtése

- egy objektumot eleve csak a metódusain keresztül illik kezelni
  - hogy a Perl megenged más is, az nagyon jól jön néha
  - például hibakeresésnél
  - Data::Dumper
- merészebb trükkökkel a reprezentáció teljesen elrejthető
  - perltoot/"Closures as objects"
- nyelvi eszköz nincs rá

50

## Implementáció elrejtése

- \_ jellel kezdődő metódusok privátnak tekintendők
- sub \_private\_method {  
    ...  
}
- a Perl megengedi, hogy meghívjuk
- nem több, mint egy konvenció

51

## POD Plain Old Documentation

52

## POD – Plain Old Documentation

- egy dokumentációleíró nyelv Perl programok dokumentálására
  - más programok dokumentálására is használható, ha külön file-ban van a dokumentáció
- egyszerű
  - nem volt cél, hogy könyvet lehessen írni vele
  - cél volt, hogy egyszerű legyen
- Perl-hez tartozó dokumentációk ebben a formában készülnek
- a feldolgozó script is Perl nyelven készült

53

## POD

- beágyazható Perl programokba
  - a dokumentációs részeket a Perl fordító figyelmen kívül hagyja
- elhelyezhető külön file-ban is
  - egy csomag mellé helyezve .pod file-ban
- perldoc parancs megkeresi és megjeleníti a dokumentációt
  - PERL5LIB környezeti változó alapján is keres
  - pod2man | nroff -man | \$PAGER
  - megadható egy konkrét file is paraméterként

54

## POD

- a dokumentáció bekezdésekre (paragraph) oszlik
- a bekezdéseket egy üres sor választja el egymástól
  - normál bekezdés
    - a szöveget betördeli, esetleg igazítja
    - formázó kódok használhatóak
      - vastag (bold)
      - dőlt (italic)
      - kód (code)
      - stb.
  - verbatim bekezdés
    - szó szerinti, ahogy van
    - minden sora szóközzel, vagy TAB karakterrel kezdődik
    - nincsenek formázó kódok

55

## POD

- bekezdések
  - parancs bekezdés
    - a formázó rendszer irányítására valók
    - többnyire egyetlen sorból állnak
  - = jellel kezdődnek
    - POD dokumentáció határai
    - fejlécek (heading)
    - listák
    - speciális dokumentáció speciális formázók részére

56

## POD

- külön bekezdések
    - azaz előtte és utána is üres sor kell
  - POD dokumentáció kezdetét jelző parancs bekezdés
- =pod
- POD dokumentáció végét jelző bekezdés
- =cut

57

## POD - fejlécek

- külön bekezdések
    - azaz előtte és utána is üres sor kell
  - =headn text
    - n: 1, 2, 3, 4 az fejléc szintje, nem pedig sorszám
    - text: a fejléc szövege
  - =head1 My Heading
- Normál bekezdés....

58

## POD – formázó kódok

- formatting codes
- I<text>
  - kiemelt (dőlt, aláhúzott, ahogy lehet)
- B<text>
  - vastagított
- L<link>
  - hyperlink
- C<code>
  - kód jelölésére
- S<text>
  - nem törölhető szöveg
- F<filename>
  - filenevek

59

## POD – formázó kódok

- E<code>
  - escape
  - E<lt>: <
  - E<gt>: >
  - E<verbar>: |
  - E<sol>: /
  - HTML kódok
    - E<acute>: é
  - karakter kódok
    - E<13>: LF

60

## POD

- `sort { $b <=> $a }`
- `C<sort { $b E<lt>=E<gt> $a }>`
- jobb megoldás
  - `C<< sort { $b <=> $a } >>`
  - szóköz a `<<` karakterek után
  - szóköz a `>>` karakterek előtt
- `open(HANDLE, ">>$file") or die;`
  - `C<<< open(HANDLE, ">>$file") or die; >>>`

61

## POD - listák

- `=over...=back` régiókban definiálunk egy listát
- `=item` jelöl minden listaelemet
- `=over n`
  - `n` karakterrel belljebb kezdődnek a listaelemek
  - `n` alapértelmezett értéke 4

`=over 4`

`=item B<my_method(>`

Description of my method.

`=back`

62

## POD - listák

- `=item *`
  - bullet lesz minden listaelem előtt
  - man oldal esetén ez egy csillag
  - HTML oldal esetén egy UL lista
- `=item 1.`
  - sorszám kerül a listaelemek elé
  - HTML oldal esetén számozott lista
- `=item ...`
  - egyéb esetek
- fontos, hogy ha egy listát valahogy elkezdünk, akkor úgy folytassuk

63

## További érdekes témák

- `tie`
  - `perldoc perltie`
- `autoload`
  - `perldoc perltoot`
  - `AutoLoader`, `SelfLoader`, `DynaLoader`
- Perl Unicode támogatása
  - `perldoc perluniintro`
  - `perldoc perlunicode`
- operátorok túlterhelése
  - `perldoc overload`
- kiíratás formátumokkal
  - `perldoc perlform`
- többszálú programok
  - `perldoc threads`, `perldoc perlthrtut`

64