

not exp log srand xor s qq qx xor  
s x x length uc ord and print chr  
ord for qw q join use sub tied qx  
xor eval xor print qq q q xor int  
eval lc q m cos and print chr ord  
for qw y abs ne open tied hex exp  
ref y m xor scalar srand print qq  
q q xor int eval lc qq y sqrt cos  
and print chr ord for qw x printf  
each return local x y or print qq  
s s and eval q s undef or oct xor  
time xor ref print chr int ord lc  
foreach qw y hex alarm chdir kill  
exec return y s gt sin sort split

not exp log srand xor s qq qx xor  
s x x length uc ord and print chr  
ord for qw q join use sub tied qx  
xor eval xor print qq q q xor int  
eval lc q m cos and print chr ord  
for qw y abs ne open tied hex exp  
ref y m xor scalar srand print qq  
q q xor int eval lc qq y sqrt cos  
and print chr ord for qw x printf  
each return local x y or print qq  
s s and eval q s undef or oct xor  
time xor ref print chr int ord lc  
foreach qw y hex alarm chdir kill  
exec return y s gt sin sort split

---

„Ezt egy egysoros programmal meg lehet oldani”  
Hatékony szkriptnyelvek Unixon 25 éve és ma

Szabó Péter <[szabo.peter@szszi.hu](mailto:szabo.peter@szszi.hu)>  
szabad szoftver tanácsadó  
Szabad Szoftver Intézet

LME GNU/Linux Szakmai konferencia 2006 diák  
Budapest, 2006-11-23 14:15–15:10

E diák szabadon felhasználható a [CC-BY-SA-2.0](https://creativecommons.org/licenses/by-sa/2.0/) licenc szerint.

# Szkriptnyelvek 25 éve és ma

Morzsák

Szkriptnyelvek 25  
éve és ma

LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes

konvertálása

Játék a Perllel

Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

25 éve is megvoltak :

Bourne shell	1977–
AWK	1977
C shell	1979
<i>Make</i>	1977
sed <sup>†</sup>	1974

Újkeletűek, aktív fejlesztés alatt állnak :

GNU ld	≤1996
<i>Lua</i>	1994–
PHP <sup>†</sup>	1995–
<i>Perl</i>	1987–
Pike <sup>†</sup>	1994–
<i>Python</i>	1990–
<i>Ruby</i>	1995–
TCL <sup>†</sup>	1988–

† csak a cikkben szerepel

*kiemelt az előadásban*

# LightTPD, mod\_magnet, Lua

```
fn=lighty.env["physical.path"]; stat=lighty.stat(fn)
print(fn) -- Dat: CGI is
if stat then
    ct=string.lower(stat["content-type"] or "")
    if ct:sub(1,5)=="text/" then
        cs=guess_charset(fn)
        if cs then
            lighty.header["Content-Type"]=ct.."; charset"..cs
            return 0
        end
    end
end
end
```

- CGI-re is meghívódik, ekkor épp üres a "content-type".
- A mod\_magnet blokkolja az egész webszerveret.
- lehetne: `lighty.content={{filename=fn}};`  
`return 200.`
- `print(fn)` a hibanaplóba menne, megfelelő prefixszel.

Morzsák

Szkriptnyelvek 25  
éve és ma

LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes

konvertálása

Játék a Perllal

Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

# Hordozható shellszkriptek

```
#!/bin/sh
```

```
eval '(exit $?0)' && eval '#\
```

```
echo Csak Bourne-kompatibilis shellben, pl. bash; exit'
```

```
echo Csak C shellben
```

- Az 1980-as években egyes rendszereken `/bin/sh` nem a szokásos Bourne shell, hanem valamely C shell volt, ezért a hordozható szkripteknek fel kellett készülniük mindkét shell-re. Általában a szétválasztás után a C shelles változat elindította önmagát Bourne shellben. Jó példa hordozható shellszkriptre a Perl *Configure* szkriptje (a <http://www.cpan.org/src/stable.tar.gz>-ben).
- Bourne shell esetén a `$?` az elző parancs visszatérési kódját adja (tehát `exit 00` lesz), C shell esetén pedig `$?0` 1-et ad vissza, ha van aktív szkriptfájl (tehát `exit 1` lesz).
- Ma már a Bash elterjedtsége miatt a hordozhatósághoz gyakran elég egy `#!/bin/bash -- shebang`.

Morzsák  
Szkriptnyelvek 25  
éve és ma  
LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes  
konvertálása

Játék a Perllel  
Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

```

OUTPUT_FORMAT(binary) TARGET(elf32-little) SECTIONS {
  __myorg = 0x08048000 ; . = __myorg ;
  .text . : /*SUBALIGN(4)*/ {
    __ehdr = ABSOLUTE(.) ;
    BYTE(0x7F) BYTE(69) BYTE(76) BYTE(70) BYTE(1) BYTE(1) BYTE(1)
    /* ^^^ db 0x7F, "ELF", 1, 1, 1 ; 0 e_ident ... */
    LONG(__p_filesz - __myorg) /* ... */
    BYTE(0x89) BYTE(0x0D) LONG(environ) /* mov [environ],ecx ... */
    *(.text)
  }
  . = ALIGN(4) ; .data . : /*ALIGN(4):SUBALIGN(4)*/{ *(.data) }
  . = ALIGN(4) ; .rodata . : /*ALIGN(4):SUBALIGN(4)*/{ *(.rodata) }
  __p_filesz = . ; . = ALIGN(4) ;
  .bss . : /*ALIGN(4): SUBALIGN(4)*/ {
    environ = ABSOLUTE(.) ; . += 4 ; *(.bss) }
  __p_memsz = . ;
}

```

## Használata:

```

gcc -c -Os src*.c
ld -T tiny_reloc2.scr -o prog src*.o

```

# GNU ld linker szkriptek

- A linker szkriptek segítségével szinte tetszőleges bináris fájlformátum támogatása megvalósítható.
- A példában egy x86-os mini ELF binárist linkelünk. Ha az egészet assemblyben íránk, akkor pl. a *true* program 43 bájtos lenne – módszerünkkel 135 bájt. A `/bin/true` 12960 bájtos (!).
- A `libc` nem használható, csak rendszerhívások vannak.
- C++ programokra nem jó, mert nem kezeli az egyéb szekciókat (pl. konstruktorok, VMT).
- A példafájlok (*tinyelf*) letölthetők:  
<http://www.inf.bme.hu/~pts/>
- Bővebb infó kis ELF-ek gyártásáról assemblyben: A *Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux*.

Morzsák

Szkriptnyelvek 25 éve és ma

LightTPD, mod\_magnet, Lua

Hordozható shellszkriptek

GNU ld linker szkriptek

Önlefordító C program

Butított Makefile L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás regexekkel

C kommentek helyes

konvertálása

Játék a Perllel

Levélszűrés két sorban

Levélszűrés ékezhelyesen

GMail levelek letöltése

Iterálás

# Önlefordító C program

Morzsák  
Szriptnyelvek 25  
éve és ma  
LightTPD,  
mod\_magnet, Lua  
Hordozható  
shellszkriptek  
GNU ld linker  
szkriptek

## Önlefordító C program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz  
Szövegfeldolgozás  
regexpekkel  
C kommentek  
helyes  
konvertálása  
Játék a Perllel  
Levélszűrés két  
sorban  
Levélszűrés  
ékezhelyesen  
GMail levelek  
letöltése

Iterálás

```
#define X X /*
set -ex
CFLAGS="-O3 -s -DNDEBUG=1"; [ "$1" ] && CFLAGS="-g"
${CC:-gcc} $CFLAGS -ansi -pedantic -Wunused -Wall \
    -W -Wstrict-prototypes -Wnested-externs -Winline \
    -Wpointer-arith -Wbad-function-cast -Wcast-qual \
    -Wmissing-prototypes -Wmissing-declarations "$0" -o hw
exit #*/
#include <stdio.h>
int main(int argc, char **argv) {
    (void)argc; (void)argv;
    return 0>puts("Hello, World!");
}
```

- A kétnyelvű program alaptrükkje: az egyik nyelvben váltunk hamar kommentbe.
- A `define X X` hatástalan, nem okoz végtelen rekurziót.
- Nem shebanggel (`#!`) kezdődik, mert az C-ben hibás lenne.

```
DIRNAME=$(notdir $(shell pwd))
JOBNAME=$(DIRNAME)
MODE=draft
.PHONY: final clean
compiled.pdf: $(JOBNAME).tex $(wildcard $(JOBNAME).bib fig_*.pdf)
    $(MAKE) clean
    pdflatex '\def\OPTS{$(DIRNAME),$(MODE)}\input' \
        $(JOBNAME) </dev/null
    if grep -l '^\\citation *[\{' $(JOBNAME).aux; then \
        bibtex $(JOBNAME) </dev/null; fi
    pdflatex '\def\OPTS{$(DIRNAME),$(MODE)}\input' \
        $(JOBNAME) </dev/null
    cp $(JOBNAME).pdf compiled.pdf
final final.pdf:
    grep '^\\etproc@set@final' page1no.cfg # gyors ellenőrzés
    $(MAKE) clean
    $(MAKE) MODE=final compiled.pdf
    mv compiled.pdf final.pdf
clean:
    rm -f *.aux *.out *.bbl *.lof *.lot *.toc compiled.*
```

# Butított Makefile $\LaTeX$ -fordításhoz

Morzsák

Szkriptnyelvek 25 éve és ma

LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
 $\LaTeX$ -fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes

konvertálása

Játék a Perllel

Levélszűrés két  
sorban

Levélszűrés  
ékezet helyesen

GMail levelek  
letöltése

Iterálás

- Sajnos sok szoftvernek túl bonyolult a Makefile-struktúrája, többtucat Makefile egymást include-olja. Egy jobb modellben tömörebben, átláthatóbban lehetne fogalmazni.
- Érzékeny a tabokra (az akció sorai nem kezdődhetnek szóközzel). Ebben kicsit hasonlít a Pythonra és a Haskellre.
- Mi van, ha kettőnél több fordítás szükséges? Az `.aux`, `.toc`, `.out` stb. fájlok módosítási dátuma alapján kéne újrafordítani. Ez Make-ben nem leírható, mert körkörös függőséget tartalmaz.
- Egyéb hiány a Make-ben: a szabályok automatikus generálása nem elég rugalmas.
- A Makefile-ban a shell-programozás (`if ... then`) is megjelenik. UNIX alatt `/bin/sh`, Bourne shell feltételezhető.
- Lehetőség van a Make-változók parancssori felülbírálására (pl. `MODE=final`).

# Szövegfeldolgozás regexpekkel

Alakítsunk minden // C++ szintaxisú kommentet /\* C \*/ szintaxisúvá! Néhány egysoros megoldás:

```
sed 's@//\(.*\)\@/*\1 */@' <BE >KI
```

```
awk '{print gensub(/\//\(.*)/, "/*\1 */", 1)}' <BE >KI
```

```
perl -pi -e's@//(.*)\@/*$1 */@g' FÁJL
```

```
ruby -pi -e'$_gsub!(/\//\(.*)/, "/*\1 */")' FÁJLOK
```

```
php5 -R'echo ereg_replace("//(.*)", "/*\1 */", $argn).  
"\n";' <BE >KI
```

```
python -c 'import sys; import re  
for s in sys.stdin.xreadlines():  
    sys.stdout.write(re.sub("//(.*)", "/*\1 */", s))' <BE>KI
```

Rövidebb pythonos egysorosok írásához hasznos a *PyOne*.

Az összes fenti megoldás tartalmazza ugyanazt a hibát. Mit is?

Morzsák

Szkriptnyelvek 25  
éve és ma

LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes  
konvertálása

Játék a Perllel  
Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

# C kommentek helyes konvertálása

```
perl -0777 -pi -e's{ ( //([^\n]*) | "(?:[^\\""]+|\\\.)*" |  
  /*$2 */ : $1 }gesx' *.c
```

- A javított változat figyelmen kívül hagyja a stringen vagy kommenten belüli //-eket.
- Hasonló bonyolultságú például egy HTML vagy XML-fájl beolvasása, bizonyos tagek vagy attribútumok cseréje.
- Bonyolultabb például C forráskód értelmezése, mert az egymásba skatulyázott zárójelet tartalmazó kifejezést nem lehet regexppel leírni (de Perl regexppel igen, és Perl6-ban ez még könnyebb lesz, mert ott már lesz CF nyelvi elemző).
- Kombinált regexp- és iterátorhasználat Perlben és Rubyban is lehetséges. A *pos()*-t is igénylő igazán bonyolultakba (pl. WikiText konvertálása HTML-lé, modulárisan bővíthetően) csak Perlben érdemes nekifogni.

Morzsák

Szkriptnyelvek 25

éve és ma

LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU ld linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexpekkel

C kommentek  
helyes  
konvertálása

Játék a Perllel

Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

# Játék a Perllel

Morzsák  
Szriptnyelvek 25  
éve és ma  
LightTPD,  
mod\_magnet, Lua  
Hordozható  
shellszkriptek  
GNU ld linker  
szkriptek  
Önlefordító C  
program  
Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz  
Szövegfeldolgozás  
regexekkel  
C kommentek  
helyes  
konvertálása

## Játék a Perllel

Levélszűrés két  
sorban  
Levélszűrés  
ékezhelyesen  
GMail levelek  
letöltése

Iterálás

```
not exp log srand xor s qq qx xor
s x x length uc ord and print chr
ord for qw q join use sub tied qx
xor eval xor print qq q q xor int
eval lc q m cos and print chr ord
for qw y abs ne open tied hex exp
ref y m xor scalar srand print qq
q q xor int eval lc qq y sqrt cos
and print chr ord for qw x printf
each return local x y or print qq
s s and eval q s undef or oct xor
time xor ref print chr int ord lc
foreach qw y hex alarm chdir kill
exec return y s gt sin sort split
```

- Mike Rosulek alkotása. Részletesen kielemezve itt:  
[http://perlmonks.org/?node\\_id=290607](http://perlmonks.org/?node_id=290607)
- Egyéb perles játékok: ki tudja megírni a legrövidebben?  
(Perl golf), versírás.

# Levélszűrés két sorban

A tárgyukban unalmas-t tartalmazó levelek törlése egy mbox formátumú levelesládából:

```
awk '/^From / { if(!d&& m) print substr(m,2); m=""; b=0}
      !d && !b && (tolower($0)~/^subject: .*unalmas/) {d=1}
      !b && /^$/{b=1}
      {m = m "\n" $0}
      END{if (!d&& m) print substr(m,2)}' <BE >KI
```

Kár, hogy ékezetes betűket nem találja meg. Pl. az álmósító nézhet így ki: =?UTF-8?B?77+9bG1vc++/vXTvv70=?=. Erre bajosan illesztünk regexpet...

Először írjuk át Perlbe:

```
perl -ni -e 'if(/^From /){print$M if!$D;$M="";$B=0}
      $D||=(!$B&& /^Subject: .*unalmas/i; $B||=/^$/;
      $M.=$_; END{print$M if!$D}' MBOXFÁJL
```

Morzsák

Szkriptnyelvek 25

éve és ma

LightTPD,

mod\_magnet, Lua

Hordozható

shellszkriptek

GNU ld linker

szkriptek

Önlefordító C

program

Butított Makefile

L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás

regexpekkel

C kommentek

helyes

konvertálása

Játék a Perllel

Levélszűrés két sorban

Levélszűrés

ékezethelyesen

GMail levelek

letöltése

Iterálás

# Levélszűrés ékezethelyesen

Abban reménykedünk, hogy a CPAN-ről, a friss Perl-modulok lelőhelyéről le tudunk tölteni olyan modult, ami dekódolja a levél MimeWords formumú tárgyát. Némi keresgélés után (<http://searc.cpan.org/>) a *MIME::AltWords* modult telepítjük teljesen automatikusan: `cpan MIME::AltWords`.

```
perl -ni -e '
    sub u_lc($) { my $S=$_[0];utf8::upgrade($S);lc($S) }
    if(/^From /){print$M if!$D;$M="";$B=0}$D||=(!$B&&
    /^Subject: (.*)/i&& u_lc(MIME::AltWords::
    decode_mimewords($1, Raw=>0))=~/álmosító/);
    $B||=/^$/; $M.=$_; END{print$M if!$D}' MBOXFÁJL
```

Sajnos a Perl unicode-os regexpjei megkülönböztetik a kis és nagy ékezetes betűket, ezért előbb kisbetűsítettük (`u_lc`), és csak utána illesztettünk.

Morzsák  
Szriptnyelvek 25  
éve és ma  
LightTPD,  
mod\_magnet, Lua  
Hordozható  
shellszkriptek  
GNU Id linker  
szkriptek  
Önlefordító C  
program  
Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz  
Szövegfeldolgozás  
regexpekkel  
C kommentek  
helyes  
konvertálása  
Játék a Perllel  
Levélszűrés két  
sorban  
Levélszűrés  
ékezethelyesen  
GMail levelek  
letöltése  
Iterálás

# GMail levelek letöltése

Ismét abban bízunk, hogy valaki már megírta a szoftvert helyettünk, és feltöltötte a CPAN-re. Némi keresgélés és próbálgatás után a *Mail::Webmail::Gmail* modul mellett döntünk.

```
use Mail::Webmail::Gmail; use integer; use strict;
my $gmail = Mail::Webmail::Gmail->new(
    username => 'johndoe', password => 'secret');
die if !$gmail->check_login();
for my $conv (@{$gmail->get_messages(label=>'lepkek')}) {
    if ($conv->{ 'new' } and
        (my $msgs=$gmail->get_indv_email( msg => $conv))) {
        for my $key (sort keys %$msgs) {
            my $S=$gmail->get_mime_email(msg=>($msgs->{$key}));
            die if $gmail->error();
            print "\nFrom gmail\@localhost ".
                "Thu Jan 01 00:00:01 1970\n$S"
        }
    }
}
```

Morzsák

Szkriptnyelvek 25  
éve és ma  
LightTPD,  
mod\_magnet, Lua

Hordozható  
shellszkriptek

GNU Id linker  
szkriptek

Önlefordító C  
program

Butított Makefile  
L<sup>A</sup>T<sub>E</sub>X-fordításhoz

Szövegfeldolgozás  
regexekkel

C kommentek  
helyes  
konvertálása

Játék a Perllel

Levélszűrés két  
sorban

Levélszűrés  
ékezhelyesen

GMail levelek  
letöltése

Iterálás

# Permutációk a Ruby iterátorokkal

Fel kell sorolni egy lista összes permutációját. Rögtön adódik egy rekurzív algoritmus: válasszuk ki az első elemet (az összes lehetséges módon), majd generáljuk a maradék összes permutációját (az összes lehetséges módon). Rubyban így fest:

```
class Array
  def permutations()
    return [self.dup] if self.size<2
    ret=[]
    self.size.times { |i|
      (self[0...i]+self[i+1..-1]).permutations.each { |pe|
        ret << [self[i]]+pe
      }
    }
    ret
  end
end
```

```
p [4,5,6].permutations #: [[4,5,6],[4,6,5] ... [6,5,4]]
```

Morzsaák

Iterálás

Permutációk a  
Ruby iterátorokkal

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutínok  
Rubyban

# ... és saját iterátorral

Készítsünk saját iterátort!

```
class Array
  def each_permutation(&proc)
    if self.size<2; proc.call(self.dup); return end
    self.size.times { |i|
      (self[0...i]+self[i+1..-1]).each_permutation { |pe|
        proc.call([self[i]]+pe)
      }
    }
  end
  def permutations()
    ret=[]
    each_permutation { |pe| ret << pe }
    ret
  end
end
```

```
[4,5,6].each_permutation { |pe| p pe }
p [:a,:b,:c,:d,:e,:f].permutations.size #: 720
```

Morzsák

Iterálás

Permutációk a  
Ruby iterátorraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Permutációk Perlben

A ruby-s megoldást betűről betűre le tudjuk fordítani. Kár, hogy közben elszabadulnak a @ \$ { [ ( ) ] } jelek

```
sub each_permutation($$) {
    my($ary,$proc)=@_;
    if (@$ary<2) { $proc->([@$ary]); return }
    for (my $I=0; $I<@$ary; $I++) { each_permutation(
        [@$ary[0..$I-1],@$ary[$I+1..$#$ary]],
        sub { $proc->([$ary->[$I], @{$_ [0]}]) });
    }
}
sub permutations($) {
    my @ret;
    each_permutation($_[0], sub { push @ret, $_[0] });
    @ret
}

each_permutation([qw(a b c)], sub{ print"@{$_ [0]}\n" });
print permutations([qw(a b c d e f)])."\n"; #: 720
```

Morzsák

Iterálás

Permutációk a  
Ruby iterátorraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Iterálás, zárvány

- Egy gyűjtemény elemein való végigmenést (például lista elemein, fájl sorain, osztály metódusain) nevezzük iterálásnak. A legtöbb nyelv tartalmaz erre beépített eszközt (pl. Ruby iterátorok, C++ iterátorok).
- Ha a nyelvben (pl. Perl) nincs külön eszköz, akkor zárványokkal könnyen mevalósítható.
- A zárvány (*lexical closure*) egy olyan belső függvény (`$proc`), amely hozzáfér a külső függvény lokális változóihoz (`$ary`, `$I`). Plusz funkció még: a belső függvény a külső függvény visszatérte után is meghívható.
- Megjegyzés: zárvány Javában is van, például anonim osztályokkal implementálható:

```
new Runnable() { public void run() { ... } }
```

Morzsák

Iterálás

Permutációk a  
Ruby iterátoraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Permutációk Perlben – a csúnya út

```
sub perm { @_<2 ? [ @_ ] : map { my $I=$_; map { [$_[$I],  
    @$_ ] } perm(@_[0..$I-1],@_[$I+1..$#_]) } 0..$#_ }
```

```
for my $T (@{ perm(qw(a b c)) }) { print "$T.\n" }
```

- Az Igazi Perl-programozó nem engedi, hogy a kódban a betűk túlsúlyba kerüljenek... Ő sem akarja megérteni mások kódját – akkor mások miért értsék az övét?
- Szerencsére a CPAN-es modulokat nem a csúnya utat járó Igazi Programozók írják.
- „Több út van (*There's more than one way to do it.*)” – így szól a Perl egyik jelmondata. A másik: nincsenek szükségtelen korlátozások (*No unnecessary limits*).
- A Perl-t szokták svájcbicskához is hasonlítani: minden funkció megvan benne, melyek közül okosan kell választani.
- A Perl 6 nyelvi szinten még ennél is tovább megy majd.

Morzsák

Iterálás

Permutációk a  
Ruby iterátorraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Permutációk Pythonban

Morzsák

Iterálás

Permutációk a  
Ruby iterátorraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

```
def gen_permutations(ary):
    if len(ary)<2: yield ary[:]; return
    for i in xrange(len(ary)):
        for pe in gen_permutations(ary[0:i]+ary[i+1:]):
            yield [ary[i]]+pe

def permutations(ary):
    ret=[]
    for pe in gen_permutations(ary): ret+= [pe]
    return ret

print permutations(['a','b','c'])
print len(list(gen_permutations(list(xrange(6))))) #:720
```

- Nincs `end`, mindent a beljebb kezdés dönt el.
- A `for` szintaxisa miatt is jobban olvasható, mint a Ruby-változat.

# Prímszámok Python generátorokkal

Morzsák

Iterálás

Permutációk a  
Ruby iterátoraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

Eratoszthenészi szita (*sieve*): vedd a számokat 2-től épnek, írd ki legkisebb ép számot, lyukaszd ki a többszöröseit (vagyis szüntesd meg az épségüket), folytasd a végtelenségig.

```
import sys; sys.setrecursionlimit(999999999)
def intsfrom(i):
    while 1:
        yield i
        i = i + 1
def exclude_multiples(n, ints):
    for i in ints:
        if (i % n):
            yield i
def sieve(ints):
    while 1:
        prime = ints.next()
        yield prime
        ints = exclude_multiples(prime, ints)

for prime in sieve(intsfrom(2)): print prime
```

# A generátorokról

Morzsák

Iterálás

Permutációk a  
Ruby iterátoraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

- *generátor*: adatelemeket generáló függvény, amely korutin-ként felváltva fut a főprogrammal, és a főprogram *next()*-tel kérheti a következő adatelemet.
- A prímszámgenerálós példán kívül egyéb generátoros trükkök: <http://linuxgazette.net/100/pramode.html>
- A *Stackless Python* egyáltalán nem használja a C vermet. Ez egyébként kezel mikroszálakat (szinkronizációval) és *call/cc*-t is. E területeken gyorsabb a sima Pythonnál. Pl. 30000-ig a prímeket 1.55-ször gyorsabban találta meg a példabeli programmal.
- Egyéb lehetőségek a végrehajtás késleltetésére: *Generalization of Deferred Execution in Python*.
- Iterálásról, generátorokról, *call/cc* Scheme nyelven: <http://okmij.org/ftp/Scheme/enumerators-callcc.html>

# Prímgenerálás Rubyban

A Pythonos megoldás szóról szóra lefordítható Rubyra:

```
require 'pts_generator.rb' # http://www.inf.bme.hu/~pts/
def mkgener(obj,methodname,*args)
  Generator.new { |g| obj.send(methodname,*args,
    &g.method(:yield)) }
end
def intsfrom(i) while true; yield i; i+=1; end end
def exclude_multiples(n, ints)
  ints.each { |i| yield i if 0!=(i%n) }
end
def sieve(ints)
  while prime=ints.next();
    yield prime
    ints=mkgener(self, :exclude_multiples, prime, ints)
  end
end

sieve(mkgenerator(2..23, :each)) { |prime| p prime }
sieve(mkgenerator(self, :intsfrom, 2)) { |prime| p prime }
```

Morzsák

Iterálás

Permutációk a  
Ruby iterátorával

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Iterátor és generátor Rubyban

Morzsák

Iterálás

Permutációk a  
Ruby iterátoraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

- Sajnos a fenti megoldás pazarló: 10000-ig az 1229 db prímszám megtalálása Rubyban 316.85s (elfogyasztott kb. 500 MB memóriát), ez a Python idejének kb. 422-szerese. A példa is mutatja, hogy Rubyban a korutinok és a generátorok nem hatékonyak.
- *iterátor*: adatelemeket generáló függvény, amely egy előre megadott zárványt hív minden elem generálásakor.
- A Ruby népszerűségét részben az iterátoroknak is köszönheti. Pontosabban annak, hogy a standard függvénykönyvtár sok iterátort kínál, melyeket a programozó könnyen és tömören tud kombinálni.
- Az iterátorok kevesebbet tudnak, mint a generátorok (például nem lehet egy végtelen iterátortól az első néhány elemet kérni – ez kell a prímszításhoz is). Rubyban az generátorokat iterátorokkal és korutinokkal emulálják (mint mi a prímszítás példában).

# Két játékos egy programban

Játsszunk gyufaszálasat! Ketten lépnek felváltva. Egy lépésben 1-et, 2-t vagy 3-at lehet elvenni. Az nyer, aki elveszi az utolsót.

```
def okoska(masik_co)
  while $asztalon>0;
    $asztalon-=el=($asztalon%4==0)?rand(3)+1:$asztalon%4
    print "Okoska elvesz #{el}-t, marad #{$asztalon}.\n"
    masik_co.call if $asztalon>0
  end
  print "Okoska nyert.\n"; exit
end

def butuska(masik_co)
  while $asztalon>0;
    $asztalon-=el=rand(3)+1
    print "Butuska elvesz #{el}-t, marad #{$asztalon}.\n"
    masik_co.call if $asztalon>0
  end
  print "Butuska nyert.\n"; exit
end
```

Morzsák

Iterálás

Permutációk a  
Ruby iterátoraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python

generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutinok  
Rubyban

# Korutink Rubyban

Morzsák

Iterálás

Permutációk a  
Ruby iterátorraival

... és saját  
iterátorral

Permutációk  
Perlben

Iterálás, zárvány

Permutációk  
Perlben – a  
csúnya út

Permutációk  
Pythonban

Prímszámok  
Python  
generátorokkal

A generátorokról

Prímgenerálás  
Rubyban

Iterátor és  
generátor  
Rubyban

Két játékos egy  
programban

Korutink  
Rubyban

```
def mkco(&proc)
  callcc { |context| return context }
  proc.call
end
$asztalon=24; print "#{$asztalon} gyufaszál van.\n"
```

```
okoska_co=butuska_co=nil
okoska_co =mkco { okoska(butuska_co) }
butuska_co=mkco { butuska(okoska_co) }
okoska_co.call # Okoska kezd
```

- `callcc` (`call/cc`, *Call with Current Continuation*): térj át a megadott zárvány végrehajtására, és tedd lehetővé a későbbi, bármikori visszatérést a `call/cc` utánra.
- Harmadik játékos felvétele: stratégia implementálása, +1 `mkco()`-s sor, helyes körbeláncolás.
- Rubyban a generátorokat `callcc`-vel valósítja meg a `generator.rb` és a `pts_genetator.rb` is.

