1 ლექცია

PERL-ის ზოგადი მიმოხილვა

PERL-ის დაყენება

perl [**-sTtuUWX**] [**-hv**] [**-V**[:*configvar*]] [**-cw**] [**-d**[**t**][:*debugger*]] [**-D**[*number/list*]] [**-pna**] [**-F***pattern*] [**-l**[*octal*]] [**-0**[*octal/hexadecimal*]] [**-I***dir*] [**-m**[**-**]*module*] [**-M**[**-**]*'module...'*] [**-f**] [**-C [*number/list*]**] [**-P**] [**-S**] [**-x**[*dir*]] [**-i**[*extension*]] [**-e** *'command'*] [**--**] [*programfile*] [*argument*]...

Perl -e ‘ print “Hello” ; ‘

STDIN STDOUT STDERR

print დესკრიპტორი სია
print სია
print

printf

Perl –e “print __LINE__;”
perl -e “print __FILE__;”
Print “Hello!\n” x5;

\’ \” \\ \$ \@ \e \t \v

\n \r \f \b \a \033 \x1b \c[

```
Print  << EOD;
This
Is
a
“here”
Document.
EOD
```

სკალარული ცვლადები და სიები

```
$scalar1 = 1;
$scalar2 =  “Hello there!”;
Print  “H”, “e”,”l”, “l”,”o”;
Hello
$   %   @   &   *

($a, $b) = (1, 2, 3);
($a, $b) = map (lc, A, B);
```

```
%hash = ();
$hash{fruit} =apple;
$hash{sandwich} = hamburger;
$hash{drink} = bubbly;

%hash = (fruit , apple , sandwich, hamburger,
drink, bubby);
Print “$hash{fruit}\n”;
Apple

%hash = (fruit => apple , sandwich => hamburger,
Drink => bubby);
```

ლექცია 2
ოპერატორები

თერმი

სიის მარჯვენა	ოპერატორი	მარცხენა
	->	მარცხენა
	++	
	--	განუსაზღვრელია
	**	

!	
~	
\	
ერთადგილიანი +	
ერთადგილიანი -	მარჯვენა
!=	
=~	მარცხენა
*	
/	
%	
X	მარცხენა

+

,

-

მარცხენა

<<

>>

მარცხენა

<>

<=

>=

lt

gt

le

არაა განსაზღვრული

==

!=

<= >

eq

ne

Cmp არაა განსაზღვრული

& მარცხენა

|

^ მარცხენა

&& მარცხენა

	მარცხენა
..	
...	არაა განსაზღვრული
?:	მარჯვენა
=	
+=	
-=	
*=	
/=	
%=	

&=

|=

^=

.=

X=

**=

<<=

>>=

&&=

||=

მარჯვენა

‘

=> მარცხენა

not მარჯვენა

and მარცხენა

or

xor მარცხენა

პირობითი ოპერატორები და ციკლები

if (<ლოგიკური გამოსახულება>) {<ოპერატორები>} else {<ოპერატორები>}

while (<ლოგიკური გამოსახულება>) {<ოპერატორები>}

for (<საწყისი მნიშვნელობა>; <ლოგიკური გამოსახულება>; <საბოლოო მნიშვნელობა>) {<ოპერატორები>}

foreach <ჩამოთვლა> {<ოპერატორები>}

Last, next, continue, redo

მე-3 ლექცია

ციკლის ოპერატორი until

[<ჭდე>] until (<გამოსახულება>) {<ბლოკი>}

[<ჭდე>] until (<გამოსახულება>) {<ბლოკი>}
continue {<ბლოკი>}

მოდიფიკატორები
if <გამოსახულება>
unless <გამოსახულება>
while <გამოსახულება>
until <გამოსახულება>

for <გამოსახულება>
foreach <გამოსახულება>

do {<ბლოკი>}
do {<გამოსახულება>}
do <ქვეპროგრამა> (<პარამეტრები>)

goto <ჭდე>
goto <გამოსახულება>
goto <ქვეპროგრამა>
exit die

რეგულარული გამოსახულებები

m/... / შეთანადება
s/.../.../ ჩასმა
tr/.../.../ ტრანსლაცია

ცალკეული ნიშნები
ნიშანთა კლასები
ალტერნატული ნიმუშები
ქვანტიფიკატორები
წარმოსახვითი ნიშნები
მითითება ნაპოვნ ტექსტზე
დამატებითი კონსტრუქციები

\077 რვაობითი რიცხვი

\a ზარი

\d ეთანადება ციფრს

\D ნებისმიერი ნიშანი ციფრს გარდა

\E \L \U \Q ბრძანებათა მოქმედების დამთავრება

\l შემდეგი პატარა ასოა

\L შემდეგი დიდი ასოა \E -მდე

\s ეთანადება ხარვეზის ნიშნის მსგავსს

\S ეთანადება ნებისმიერ ნიშანს გარდა ხარვეზის ნიშნის მსგავსისა

\u შემდეგი არის ზედა რეგისტრის ნიშანი

\U შემდეგი არის ზედა რეგისტრის ნიშანი \E-მდე

\w ასო-ციფრული ნიშანი

\W გარდა ასო-ციფრული ნიშნისა

\x1B თექვსმეტობითი რიცხვი

მე=4 ლექცია
რეგულარული გამოსახულებები

```
if ("Hello World" =~ /World/) { print "It matches\n"; } else { print "It doesn't match\n"; }

if ("Hello World" !~ /World/) { print "It doesn't match\n"; } else { print "It matches\n"; }

$greeting = "World"; if ("Hello World" =~ /$greeting/) { print "It matches\n"; } else { print "It doesn't match\n"; }
```

```
$_ = "Hello World";
 if (/World/) { print "It matches\n"; } else { print "It
doesn't match\n"; }

"Hello World" =~ m!World!; # matches
 "Hello World" =~ m{World}; # matches
 "/usr/bin/perl" =~ m"/perl"; # matches

"Hello World" =~ /world/; # doesn't match "Hello
World" =~ /o W/; # matches "Hello World" =~ /oW/;
 # doesn't match
 "Hello World" =~ /World /; # doesn't match
```

```
"Hello World" =~ /o/; # matches 'o' in 'Hello' “
That hat is red" =~ /hat/; # matches 'hat' in 'That‘

{}[]()^$.|*+?\

"2+2=4" =~ /2+2/; # doesn't match
"2+2=4" =~ /2\+2/; # matches
"The interval is [0,1)." =~ /[0,1)./ # is a syntax error!
"The interval is [0,1)." =~ /\[0,1\)\./ # matches
"#!/usr/bin/perl" =~ /#!\/usr\/bin\/perl/; # matches

"#!/usr/bin/perl" =~ m!#\!/usr/bin/perl!;
 # easier to read
```

```
'C:\WIN32' =~ /C:\\WIN/; # matches

"1000\t2000" =~ m(0\t2) # matches
 "1000\n2000" =~ /0\n20/ # matches
 "1000\t2000" =~ /\000\t2/ # doesn't match

$foo = 'house'; 'housecat' =~ /$foo/; # matches
'cathouse' =~ /cat$foo/; # matches
'housecat' =~ /${foo}cat/; # matches
```

"housekeeper" =~ /keeper/; # matches
"housekeeper" =~ /^keeper/; # doesn't match
"housekeeper" =~ /keeper$/; # matches
"housekeeper\n" =~ /keeper$/; # matches

"keeper" =~ /^keep$/; # doesn't match "keeper" =~ /^keeper$/; # matches "" =~ /^$/; # ^$ matches

```
"dogbert" =~ /bert/;
# matches, but not what you want
 "dilbert" =~ /^bert/; # doesn't match, but ..
"bertram" =~ /^bert/;
 # matches, so still not good enough
 "bertram" =~ /^bert$/; # doesn't match, good
"dilbert" =~ /^bert$/; # doesn't match, good
 "bert" =~ /^bert$/; # matches, perfect
```

კლასების გამოყენება

/cat/; # matches 'cat‘
/[bcr]at/; # matches 'bat, 'cat', or 'rat'
/item[0123456789]/;
matches 'item0' or ... or 'item9'
"abc" =~ /[cab]/; # matches 'a'

/[yY][eE][sS]/; # match 'yes' in a case-insensitive way
'yes', 'Yes', 'YES', etc.

-]\^$

```
/[\]c]def/;
# matches ']def' or 'cdef' $x = 'bcr';
/[$x]at/; # matches 'bat', 'cat', or 'rat‘
 /[\$x]at/; # matches '$at' or 'xat‘
 /[\\$x]at/; # matches '\at', 'bat, 'cat', or 'rat'

/item[0-9]/; # matches 'item0' or ... or 'item9‘
 /[0-9bx-z]aa/;
# matches '0aa', ..., '9aa', # 'baa', 'xaa', 'yaa', or 'zaa'
/[0-9a-fA-F]/; # matches a hexadecimal digit
 /[0-9a-zA-Z_]/; # matches a "word"
#character, # like those in a Perl variable name
```

/[^a]at/; # doesn't match 'aat' or 'at', but matches
 # all other 'bat', 'cat, '0at', '%at', etc.
/[^0-9]/; # matches a non-numeric character
 /[a^]at/; # matches 'aat' or '^at'; here '^' is ordinary
\d matches a digit, not just [0-9] but also digits from non-roman scripts
\s matches a whitespace character, the set [\ \t\r\n\f] and others
\w matches a word character (alphanumeric or _), not just [0-9a-zA-Z_] but also digits and characters from non-roman scripts
\D is a negated \d; it represents any other character than a digit, or [^\d]

\S is a negated \s; it represents any non-whitespace character [^\s]
\W is a negated \w; it represents any non-word character [^\w]
The period '.' matches any character but "\n" (unless the modifier //s is in effect
/\d\d:\d\d:\d\d/; # matches a hh:mm:ss time format
/[\d\s]/; # matches any digit or whitespace character
/\w\W\w/; # matches a word char, followed by a # non-word char, followed by a word char
/..rt/; # matches any two chars, followed by 'rt‘
/end\./; # matches 'end.‘
/end[.]/; # same thing, matches 'end.'

```
$x = "Housecat catenates house and cat";
$x =~ /cat/; # matches cat in 'housecat'
$x =~ /\bcat/; # matches cat in 'catenates'
$x =~ /cat\b/; # matches cat in 'housecat‘
 $x =~ /\bcat\b/; # matches 'cat' at end of string

"" =~ /^$/; # matches
"\n" =~ /^$/; # matches, $ anchors before "\n"
"" =~ /./; # doesn't match; it needs a char
"" =~ /^.$/; # doesn't match; it needs a char
"\n" =~ /^.$/;
# doesn't match; it needs a char other than "\n“
 "a" =~ /^.$/; # matches "a\n" =~ /^.$/;
```

```
$x = "There once was a girl\nWho programmed in
Perl\n";
$x =~ /^Who/;
# doesn't match, "Who" not at start of string
$x =~ /^Who/s;
# doesn't match, "Who" not at start of string
$x =~ /^Who/m;
 # matches, "Who" at start of second line
 $x =~ /^Who/sm;
# matches, "Who" at start of second line
 $x =~ /girl.Who/; # doesn't match, "." doesn't match
"\n" $x =~ /girl.Who/s; # matches, "." matches "\n“
 $x =~ /girl.Who/m; # doesn't match, "." doesn't
```

```
$x = "There once was a girl\nWho programmed in
Perl\n";
$x =~ /^Who/m;
 # matches, "Who" at start of second line
$x =~ /\AWho/m;
# doesn't match, "Who" is not at start of string
$x =~ /girl$/m; # matches, "girl" at end of first line
 $x =~ /girl\Z/m;
# doesn't match, "girl" is not at end of string
$x =~ /Perl\Z/m;
# matches, "Perl" is at newline before end
 $x =~ /Perl\z/m;
# doesn't match, "Perl" is not at end of string
```

```
"cats and dogs" =~ /cat|dog|bird/; # matches "cat"
"cats and dogs" =~ /dog|cat|bird/; # matches "cat"

"cats" =~ /c|ca|cat|cats/; # matches "c"
"cats" =~ /cats|cat|ca|c/; # matches "cats"

"cab" =~ /a|b|c/ # matches "c" # /a|b|c/ == /[abc]/

/(a|b)b/; # matches 'ab' or 'bb‘
 /(ac|b)b/; # matches 'acb' or 'bb'
/(^a|b)c/;
# matches 'ac' at start of string or 'bc' anywhere
/(a|[bc])d/; # matches 'ad', 'bd', or 'cd‘
```

```
/house(cat|)/; # matches either 'housecat' or 'house'
/house(cat(s|)|)/;
 # matches either 'housecats' or 'housecat' or
 # 'house'. Note groups can be nested.
/(19|20|)\d\d/;
 # match years 19xx, 20xx, or the Y2K problem, xx
"20" =~ /(19|20|)\d\d/;
# matches the null alternative '()\d\d',
 # because '20\d\d' can't match

"abcde" =~ /(abd|abc)(df|d|de)/;
```

```
# extract hours, minutes, seconds
if ($time =~ /(\d\d):(\d\d):(\d\d)/) {
 # match hh:mm:ss format
$hours = $1;
$minutes = $2;
$seconds = $3; }

# extract hours, minutes, seconds
($hours, $minutes, $second) = ($time =~
/(\d\d):(\d\d):(\d\d)/);

/(ab(cd|ef)((gi)|j))/;
  1   2          34
```

```
/\b(\w\w\w)\s\1\b/;
```

me-5 leqcia

```
$a99a = '([a-z])(\d)\2\1';
 # matches a11a, g22g, x33x, etc.

$line = "code=e99e"; if ($line =~
/^(\w+)=$a99a$/){
 # unexpected behavior!
print "$1 is valid\n"; } else { print "bad line:
'$line'\n"; }

$a99a = '([a-z])(\d)\g{-1}\g{-2}'; # safe for being
interpolated
```

```
$fmt1 = '(?<y>\d\d\d\d)-(?<m>\d\d)-
(?<d>\d\d)'; $fmt2 =
'(?<m>\d\d)/(?<d>\d\d)/(?<y>\d\d\d\d)'; $fmt3 =
'(?<d>\d\d)\.(?<m>\d\d)\.(?<y>\d\d\d\d)'; for my
$d qw( 2006-10-21 15.01.2007 10/31/2005 ){ if
( $d =~ m{$fmt1|$fmt2|$fmt3} ){ print
"day=$+{d} month=$+{m} year=$+{y}\n"; } }

if ( $time =~ /(\d\d|\d):(\d\d)|(\d\d)(\d\d)/ ){
# process hour and minute
}
```

```
if ( $time =~ /(?|(\d\d|\d):(\d\d)|(\d\d)(\d\d))\s+([A-
Z][A-Z][A-Z])/ ){ print "hour=$1 minute=$2
zone=$3\n"; }

$x = "Mmm...donut, thought Homer"; $x =~
/^(Mmm|Yech)\.\.\.(donut|peas)/; # matches
foreach $expr (1..$#-) { print "Match $expr:
'${$expr}' at position ($-[$expr],$+[$expr])\n"; }

$x = "the cat caught the mouse"; $x =~ /cat/;
 # $` = 'the ', $& = 'cat', $' = ' caught the mouse‘
 $x =~ /the/; # $` = '', $& = 'the',
$' = ' cat caught the mouse'
```

```
$fmt1 = '(?<y>\d\d\d\d)-(?<m>\d\d)-(?<d>\d\d)';
$fmt2 = '(?<m>\d\d)/(?<d>\d\d)/(?<y>\d\d\d\d)';
$fmt3 = '(?<d>\d\d)\.(?<m>\d\d)\.(?<y>\d\d\d\d)';
for my $d qw( 2006-10-21 15.01.2007 10/31/2005
){ if ( $d =~ m{$fmt1|$fmt2|$fmt3} ){ print
"day=$+{d} month=$+{m} year=$+{y}\n"; } }

if ( $time =~ /(?|(\d\d|\d):(\d\d)|(\d\d)(\d\d))\s+([A-
Z][A-Z][A-Z])/ ){ print "hour=$1 minute=$2
zone=$3\n"; }
```

In addition to what was matched, Perl (since 5.6.0) also provides the positions of what was matched as contents of the @- and @+ arrays. $-[0] is the position of the start of the entire match and $+[0] is the position of the end. Similarly, $-[n] is the position of the start of the $n match and $+[n] is the position of the end. If $n is undefined, so are $-[n] and $+[n]. Then this code

```
$x = "Mmm...donut, thought Homer"; $x =~ /^(Mmm|Yech)\.\.\.(donut|peas)/; # matches
foreach $expr (1..$#-) { print "Match $expr: '${$expr}' at position ($-[$expr],$+[$expr])\n"; }
```

$x = "the cat caught the mouse"; $x =~ /cat/; # $` = 'the ', $& = 'cat', $' = ' caught the mouse' $x =~ /the/; # $` = '', $& = 'the', $' = ' cat caught the mouse‘

ლექცია 6
ქვეპროგრამები

sub NAME BLOCK
sub NAME(PROTO) BLOCK
sub NAME : ATTRS BLOCK
sub NAME(PROTO) : ATTRS BLOCK

ანონიმური ქვეპროგრამები

$subref = sub BLOCK;
$subref = sub (PROTO) BLOCK;
$subref = sub : ATTRS BLOCK;
$subref = sub (PROTO) : ATTRS BLOCK;

ქვეპროგრამების იმპორტირება

use MODULE qw(NAME1 NAME2 NAME3);

ქვეპროგრამის გამოძახება

NAME(LIST);
NAME LIST;
 &NAME(LIST);
&NAME;

```
sub max { my $max = shift(@_);
foreach $foo (@_)
 { $max = $foo if $max < $foo; } return $max; }
$bestday = max($mon,$tue,$wed,$thu,$fri);
```

```
sub get_line { $thisline = $lookahead;
 # global variables!
 LINE: while (defined($lookahead = <STDIN>))
 { if ($lookahead =~ /^[ \t]/)
 { $thisline .= $lookahead; } else { last LINE; } }
return $thisline; }
 $lookahead = <STDIN>; # get first line
while (defined($line = get_line())) { ... }

upcase_in($v1, $v2); # this changes $v1 and $v2
sub upcase_in { for (@_) { tr/a-z/A-Z/ } }
```

```
($v3, $v4) = upcase($v1, $v2);
 # this doesn't change $v1 and $v2
sub upcase { return unless defined wantarray;
# void context, do nothing
my @parms = @_;
for (@parms) { tr/a-z/A-Z/ }
return wantarray ? @parms : $parms[0]; }
```

```
&foo(1,2,3); # pass three arguments

 foo(1,2,3); # the same

 foo(); # pass a null list

&foo(); # the same

&foo; # foo() get current args, like foo(@_) !!

 foo; # like foo() IFF sub foo predeclared, else "foo“
```

```
my $foo; # declare $foo lexically

 local my (@wid, %get); # declare list of variables

local my $foo = "flurp"; # declare $foo lexical, and init
#it

 my @oof = @bar; # declare @oof lexical, and init it

my $x : Foo = $y;
```

```
local $foo; # make $foo dynamically local

local (@wid, %get); # make list of variables local

local $foo = "flurp"; # make $foo dynamic, and init
#it
local @oof = @bar; # make @oof dynamic, and
#init it
local $hash{key} = "val"; # sets a local value for this
#hash entry

 local ($cond ? $v1 : $v2);
```

local *FH; # localize $FH,

@FH, %FH, &FH ...

local *merlyn = *randal; # now $merlyn is really

$randal, plus # @merlyn is really @randal, etc

local *merlyn = 'randal'; # SAME THING: promote #'randal' to

*randal local *merlyn = \$randal;

ლექცია 7

მითითებები

```
$a1 = 7;
$refa1 = \$a1;  # ხისტი  მითითება
print $$refa1;  # მნიშვნელობის  აღება
                # მითითებით
7
print $refa1
  #  დაიბეჭდება  SCALAR(0x<მისამართი>)
$refa1 -> $a1; # მითითება  ისრით

$refa1 = “a1”; # სიმბოლური  მითითება
```

```
@masivi = (1,2,3);
$refmasivi -> @masivi;
print $refmasivi -> [0]; # დაიბეჭდება  1

@masivi = [1,2,3]; # ანონიმური  მასივი
$refmasivi =  \@masivi;
print $refmasivi -> [0]; # დაიბეჭდება  1

$ref =\\\\“hello”;
print $$$$$ref; # დაიბეჭდება  hello

$subref = sub  {print “hello”};
&$subref;          # დაიბეჭდება  hello
```

```
$a1 = 7;
$scalref = *a1{SCALAR};
print $$scalref;                    # დაბეჭდავს  7

$a1 = 0;
$a1sax = “a1”;
$$a1sax = 7;
print “$a1”;                        # დაიბეჭდება  7
```

```
@misalmeba = (“გამარჯობა”, “სალამი”,
                “აქამშვიდობა”);
foreach my $term  (@misalmeba)
        {*{“print” . $term} = sub{print “$term\n”};}

printgamarjoba(); # დაბეჭდავს  -  გამარჯობა
printsalami();  #   დაბეჭდავს  -  სალამი
printaqamSvidoba(); # დაბეჭდავს  –  აქამშვიდობა

foreach my $term  (@misalmeba)
        {${“print” . $term} = sub{print “$term\n”};}
&$printgamarjoba();
```

ლექცია 8
ფაილების შეტანა და გამოტანა

```
open(INFO, "datafile") || die("can't open datafile: $!");
 open(INFO, "< datafile") || die("can't open datafile: $!");
 open(RESULTS,"> runstats") || die("can't open runstats: $!");
 open(LOG, ">> logfile ") || die("can't open logfile: $!");
```

```
open INFO, "< datafile" or die "can't open datafile:
$!";
open RESULTS,"> runstats" or die "can't open
runstats: $!";
 open LOG, ">> logfile " or die "can't open logfile: $!";

open(FILEHADLE, “tmp\\file.txt” ) or  die (“cannot
open file.text” );
while(<FILEHANDLE>)  {
        print;
}
<    >    >>   +<   +>   |   -|   |-    -  >-  >&
```

close <დესკრიპტორი>

print <დესკრიპტორი>, <სია>
print <სია>
print

write <დესკპიპტორი>
write <გამოსახულება>
write

read <დესკრიპტორი>, <სკალარი>, <სიგრძე>, <წანაცვლება>
read <დესკრიპტორი>, <სკალარი>, <სიგრძე>

readline <გამოსახულება>

getc <დესკრიპტორი>

POSIX

ლექცია 9-10

კლასები, ობიექტები, მემკვიდრეობითობა

1. კლასი არის პაკეტი
2. ობიექტი არის მონაცემთა ელემენტზე მითითება
3. მეთოდი არის კლასის ობიექტში ჩადგმული ქვეპროგრამა
4. მემკვიდრეობა არის ერთი კლასის საშუალებით ახალი კლასის შექმნა

```
package  Class1;
sub  new
{       my $self = {};
        bless($self);
        return  $self;
}
return 1;

#ობიექტის  შექმნა
use Class1;
my  $object = Class1->new();
```

კლასის და ობიექტის მეთოდები

```
$result = $object->calculate($op1, $op2);

my $object1 = Class1->new();

$object1->{DATA} = 1025;
my $data = $object1{DATA};

$ob1->setdata(1024);
my $data = $ob1->getdata();
```

მემკვიდრეობითობა

```
$rec = { name => "Jason",
         age => 23,
         peers => [ "Norbert", "Rhys", "Phineas"]
};
```

```
use Person;
$him = Person->new();
$him->name("Jason");
$him->age(23);
$him->peers( "Norbert", "Rhys", "Phineas" );
push @All_Recs, $him;
 # save object in array for later
printf "%s is %d years old.\n",
$him->name, $him->age;
 print "His peers are: ", join(", ", $him->peers),
"\n";
printf "Last rec's name is %s\n", $All_Recs[-1]-
>name;
```

```
package Person;
use strict;
sub new { my $self = {};
 $self->{NAME} = undef;
$self->{AGE} = undef;
$self->{PEERS} = [];
bless($self); # but see below
return $self; }
sub name
{ my $self = shift;
 if (@_) { $self->{NAME} = shift }
return $self->{NAME}; }
```

```
sub age
 { my $self = shift;
 if (@_) { $self->{AGE} = shift }
return $self->{AGE}; }
sub peers
{ my $self = shift;
if (@_) { @{ $self->{PEERS} } = @_ }
 return @{ $self->{PEERS} }; }
1;
```

```
sub new
{
 my $class = shift;
my $self = {};
 $self->{NAME} = undef;
 $self->{AGE} = undef;
$self->{PEERS} = [];
bless ($self, $class);
return $self;
}
```

```
sub END
{ if ($Debugging)
  { print "All persons are going away now.\n"; }
}

package Employee;
use Person;
@ISA = ("Person");
1;
```

ლექცია 11-12

დაპროგრამება ინტერნეტში

ASP

```
<%@ Language=PerlScript %>
<%
for($i=0; $i<=10; $i++) {
#
# Perl-ის ბრძანებები
#
}
%>
```

```
<SCRIPT Language=PerlScript RUNAT=Server>
for($i=0; $i<=10; $i++) {
#
# თქვენი Perl ბრძანებები
#
}
</SCRIPT>

print "Hello World";

$Response->Write("Hello World");
```

1) შევქმნათ ფაილი PerlScript.asp
2) შევიტანოთ"<HTML> Hello World </HTML>"
3) შევინახოთ ეს ფაილი და შემდეგ გამოვიძახოთ როგორც http://localhost/. IIS შემთხვევაში შეიძლება *<device>:\inetpub\wwwroot*
4) გავუშვათ ფაილი *http://localhost/PerlScript.asp*

```
<%@Language=PerlScript%>
<HTML>
<TITLE> PerlScript Test </TITLE>
<%
for($i=0; $i<=10; $i+=2) {
$Response->Write("<FONT SIZE=$i
COLOR=#000000>");
$Response->Write("Hello World! </FONT> <BR>");
}
%>
</HTML>
```

```
<HTML>
<TITLE> PerlScript Example </TITLE>
<script language=perlscript runat=server>
for($i=0; $i<=10; $i+=2) {
$Response->Write("<FONT SIZE=$i
COLOR=#000000>");
$Response->Write("Hello World! </FONT> <BR>");
}
</script>
</HTML>
```

```
<HTML>
<TITLE> PerlScript Example </TITLE>
<% @Language=PerlScript %>
<% for($i=0; $i<=10; $i+=2) {
%>
<FONT SIZE=<%=$i%> COLOR=#000000>
Hello World!
</FONT>
<BR>
<% } >%
</HTML>
```

```
# Set an application variable
#
$Application->Contents->SetProperty('Item',
'myName', 'Tobias');

# Access an application variable
#
$Application->Contents->Item('myName');
```