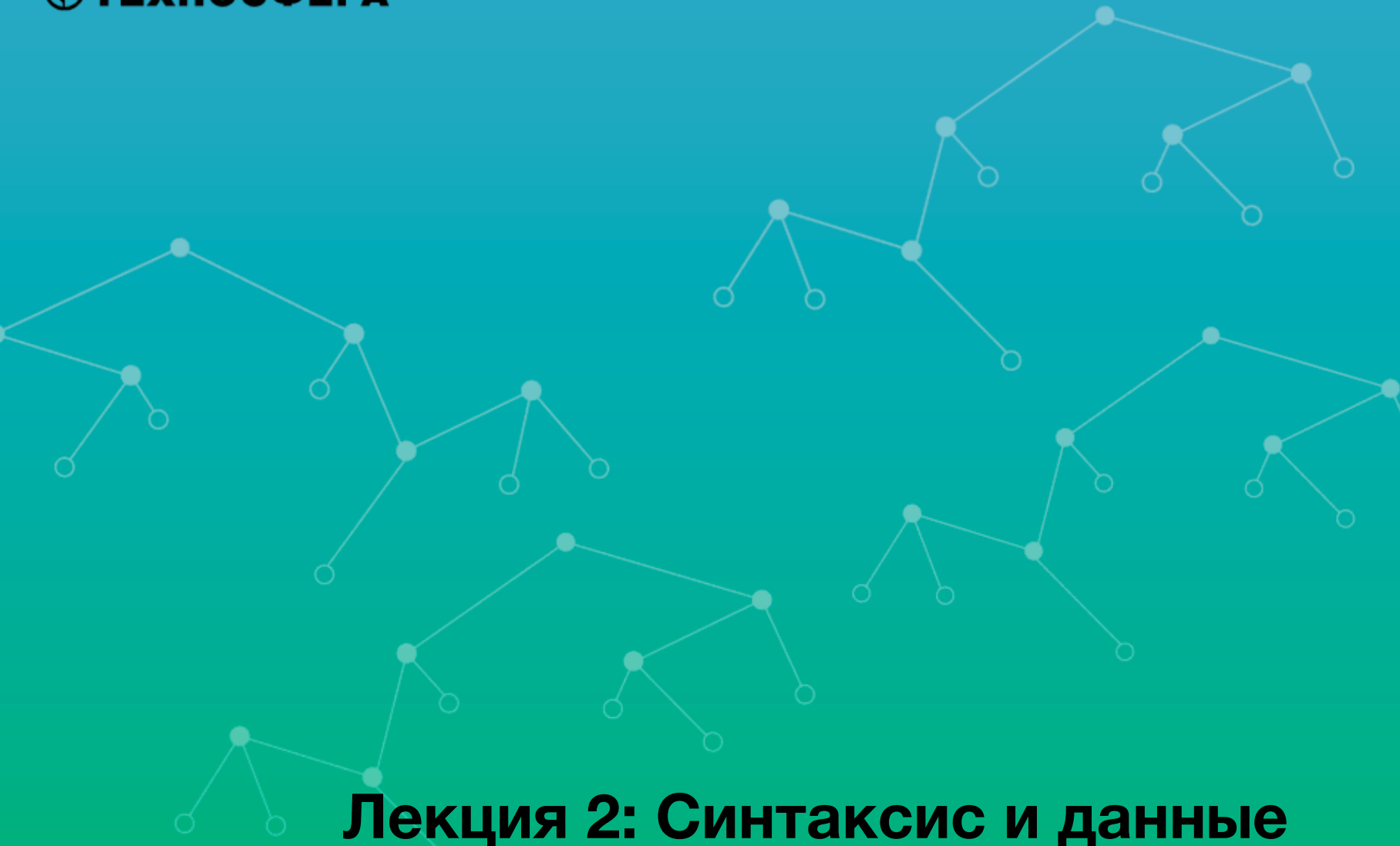


Программирование на Perl



Лекция 2: Синтаксис и данные

Содержание

- **Базовый синтаксис**
 - **Условия, циклы**
 - **Управляющие функции**
 - **Постфиксная нотация**
- **Переменные**
 - **Основные типы**
 - **Ссылки**
 - **Интерполяция**
- **Функции**
 - **Декларация, аргументы**
 - **Контекст**
 - **Прототипы**
 - **Встроенные функции**
 - **grep, map, sort**
 - **eval**
- **Операторы**
 - **Порядок исполнения**
 - **Особенные операторы**
 - **Числа и строки**

TIMTOWTDI

There's More Than One Way To Do It

**The only thing can parse
Perl (the language) is
perl (the binary)**

Блок

```
{  
    statement;  
    statement;  
    ...  
}
```

Блок

```
{  
    statement;  
    statement;  
    ...  
}
```

```
do { ... } ≠ { ... }
```

```
$value = do { ... }
```

```
$value = { ... };
```

Управление циклами

next

```
for my $item ( @items ) {  
    my $success = prepare($item);  
  
    unless ($success) {  
        next;  
    }  
  
    process($item);  
} continue {  
    # next переходит сюда  
    postcheck($item);  
}
```


Управление циклами

last

```
for my $item ( @items ) {  
    my $success = prepare($item);  
  
    unless ($success) {  
        last;  
    }  
  
    process($item);  
} continue {  
    postcheck($item);  
}  
# last переходит сюда
```

Управление циклами

redo

```
for my $item ( @items ) {  
    # redo переходит сюда  
    my $success = prepare($item);  
  
    unless ($success) {  
        redo;  
    }  
  
    process($item);  
} continue {  
    postcheck($item);  
}
```

Блок ідентичен одиночному циклу

```
{
    # redo
    stmt;
    if (...) { next; }

    stmt;
    if (...) { last; }

    stmt;
    if (...) { redo; }

    stmt;
    # next
}
# last
```

Выбор - `given` / `when`

```
use feature 'switch'; # v5.10+
```

```
given ( EXPR ) {  
    when ( undef )      { ... }  
    when ( "str" )      { ... }  
    when ( 42 )         { ... }  
    when ( [4,8,15] )   { ... }  
    when ( /regex/ )    { ... }  
    when ( \&sub )      { ... }  
    when ( $_ > 42 )    { ... }  
    default             { ... }  
}
```

Выбор - `given` / `when`

```
use feature 'switch'; # v5.10+
```

```
given ( EXPR ) {  
    when ( undef )      { ... }  
    when ( "str" )      { ... }  
    when ( 42 )         { ... }  
    when ( [4,8,15] )   { ... }  
    when ( /regex/ )    { ... }  
    when ( \&sub )      { ... }  
    when ( $_ > 42 )    { ... }  
    default             { ... }  
}
```

- Нет ключевого слова `break`
- `continue` для "проваливания"

Переход - goto

```
goto LABEL;
```

```
LABEL1:  
    say "state 1";  
    goto LABEL2;  
LABEL2:  
    say "state 2";  
    goto LABEL1;
```

```
state 1  
state 2  
state 1  
state 2  
...
```

Переход - goto

```
goto EXPR; # DEPRECATED
```

```
{  
EVEN:  
    say "even";  
    last;  
ODD:  
    say "odd";  
    last;  
}  
  
goto(  
    ("EVEN", "ODD")[ int(rand 10) % 2 ]  
);
```

Переход - goto - tailcall

```
goto &NAME;  
goto &$var;
```

```
sub fib {  
    return 0 if $_[0] == 0;  
    return 1 if $_[0] == 1;  
    return _fib($_[0]-2,0,1);  
}  
  
sub _fib { my ($n,$x,$y) = @_;  
    if ($n) {  
        return _fib( $n-1, $y, $x+$y );  
    }  
    else {  
        return $x+$y;  
    }  
}
```


Переход - goto - tailcall

```
goto &NAME;  
goto &$var;
```

```
sub fib {  
    return 0 if $_[0] == 0;  
    return 1 if $_[0] == 1;  
    return _fib($_[0]-2,0,1);  
}  
  
sub _fib { my ($n,$x,$y) = @_;  
    if ($n) {  
        @_ = ( $n-1, $y, $x+$y ); goto &_fib;  
    }  
    else {  
        return $x+$y;  
    }  
}
```

Переход - goto - tailcall

```
goto &NAME;  
goto &$var;
```

```
sub fac {  
    my $n = shift;  
    return _fac($n,1);  
}  
  
sub _fac {  
    my ($n,$acc) = @_  
    return $acc if $n == 0;  
    @_ = ($n-1,$n*$acc);  
    goto &_fac;  
}
```

Постфиксная нотация

```
STMT if EXPR;
```

```
STMT unless EXPR;
```

```
STMT while EXPR;
```

```
STMT until EXPR;
```

```
STMT for LIST;
```

```
STMT when EXPR;
```

Постфиксные циклы

```
do {  
    ...;  
} while ( EXPR );
```

```
do {  
    ...;  
} until ( EXPR );
```

```
do {  
    ...;  
} for ( LIST );
```

Постфиксные циклы

```
do {  
    ...;  
} while ( EXPR );
```

- Не работает `next`
- Не работает `last`
- Не работает `redo`
- Нет места для `continue {...}`

Постфиксные циклы

```
do {  
    ...;  
} while ( EXPR );
```

- Не работает `next`
- Не работает `last`
- Не работает `redo`
- Нет места для `continue {...}`

Нет обрамляющего блока

Содержание

- **Базовый синтаксис**
 - Условия, циклы
 - Управляющие функции
 - Постфиксная нотация
- **Переменные**
 - Основные типы
 - Ссылки
 - Интерполяция
- **Функции**
 - Декларация, аргументы
 - Контекст
 - Прототипы
 - Встроенные функции
 - `grep`, `map`, `sort`
 - `eval`
- **Операторы**
 - Порядок исполнения
 - Особенности операторы
 - Числа и строки

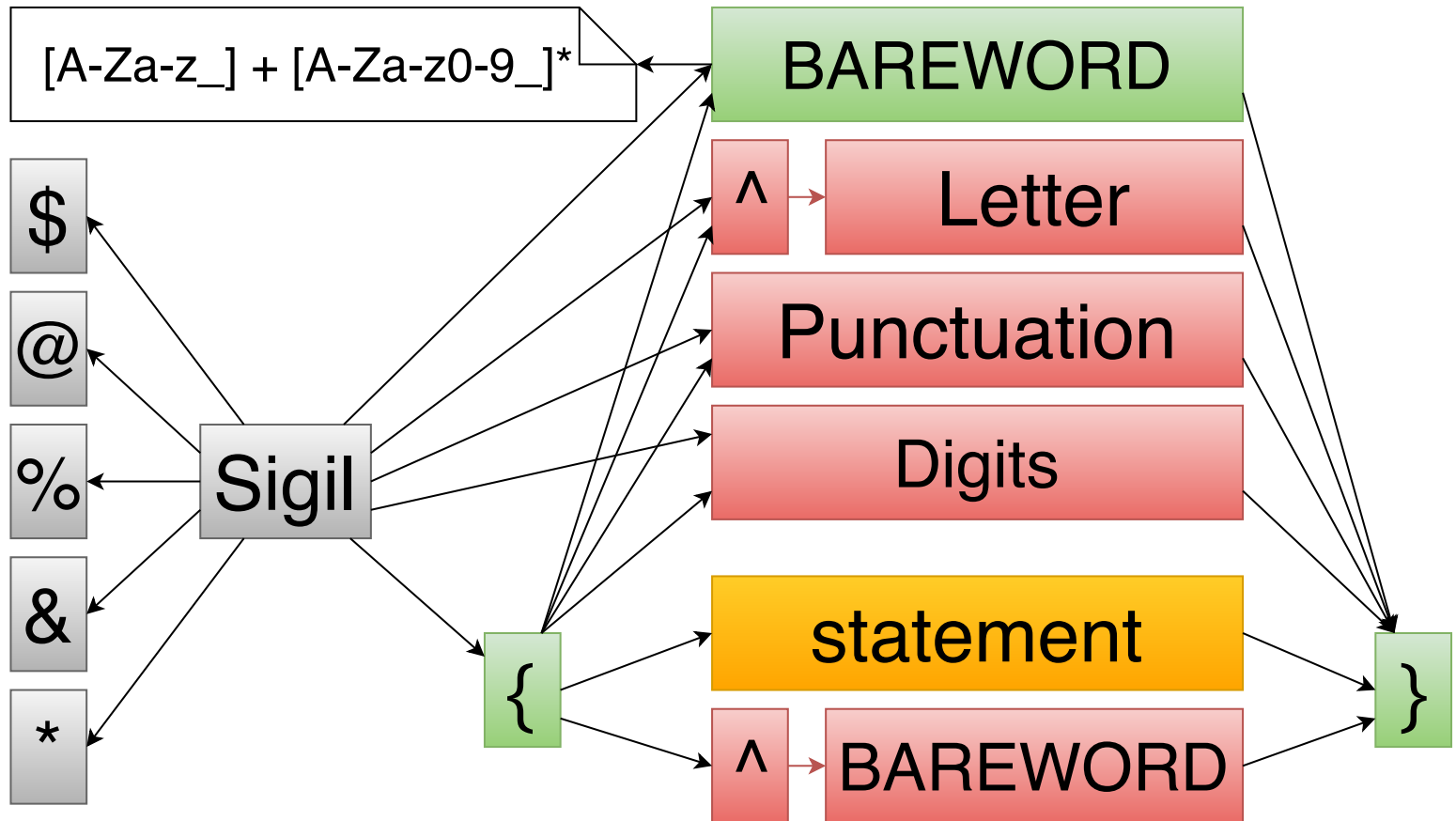
Переменные

- **SCALAR**
 - **Number**
 - **String**
 - **Reference**
- **ARRAY**
 - **Of scalars**
- **HASH**
 - **Key: string**
 - **Value: scalar**

Переменные

- **SCALAR** (`$s`)
 - **Number** (`$s = 1`, `$s = -1e30`)
 - **String** (`$s = "str"`)
 - **Reference**
 - **Scalar** (`$$r`, `${ $r }`)
 - **Array** (`@$r`, `@{ $r }`, `$r->[...]`)
 - **Hash** (`_%$r`, `%{ $r }`, `$r->{...}`)
 - **Function** (`&$r`, `&{ $r }`, `$r->(...)`)
 - **Filehandle** (`*$r`)
 - **Lvalue** (`$$r`, `${ $r }`)
 - **Reference** (`$$r`, `${ $r }`)
- **ARRAY** (`@a`, `$a[...]`)
- **HASH** (`%h`, `$h{key}`, `$h{...}`)

Переменные: идентификатор



Переменные: идентификатор

- Обычные

- `$var`, `@array`, `%hash`, `&func`, `*glob`
- `${var}`, `@{array}`, `%{hash}`, `&{func}`, `*{glob}`
- `${ "scalar" . "name" }`, `%{ "hash".$id }`

- Специальные

- `$^W`, `$^O`, `$^X`, `${^W}`, `${^O}`, `${^X}`, ...
- `$0`, `$1`, `$100`, `${0}`, `${1}`, `${100}`, ...
- `${^PREMATCH}`, `${^MATCH}`, `${^POSTMATCH}`, ...
- `$_`, `@_`, `$!`, `$@`, `$?`, `$"`, `$/`, `$,`, ...
- `${_}`, `@{_}`, `${!}`, `${@}`, `${?}`, `${"}`, `${/}`, `${,}`, ...
- Список - в [perlvar](#)

Переменные: SCALAR

Числа

```
$int      = 12345;  
$pi       = 3.141592;  
$pi_readable = 3.14_15_92_65_35_89;  
$plank    = .6626E-33;  
$hex      = 0xFEFF;  
$bom      = 0xef_bb_bf;  
$octal    = 0751;  
$binary   = 0b10010011;
```

Переменные: SCALAR

Строки

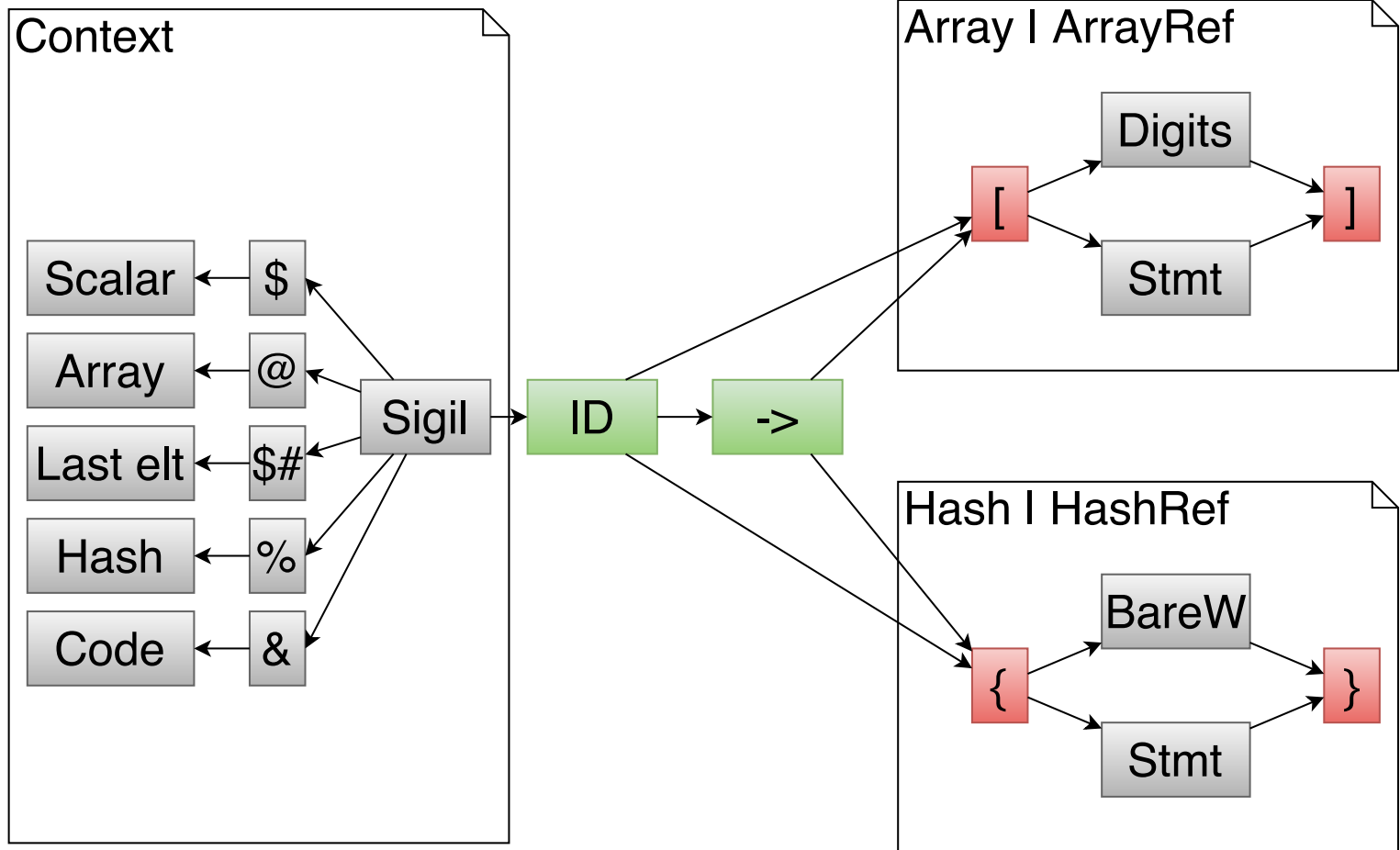
```
$one      = "string";  
$two      = 'quoted';  
$wrap     = "wrap  
          ped";  
$join     = "prefix:$one";  
$at       = __FILE__ . ':' . __LINE__;  
  
$q_1      = q/single-'quoted'/;  
$qq_2     = qq(double-"quoted"-$two);  
$smile    = ":) -> \x{263A}";  
$ver      = v1.2.3.599;
```

Переменные: SCALAR

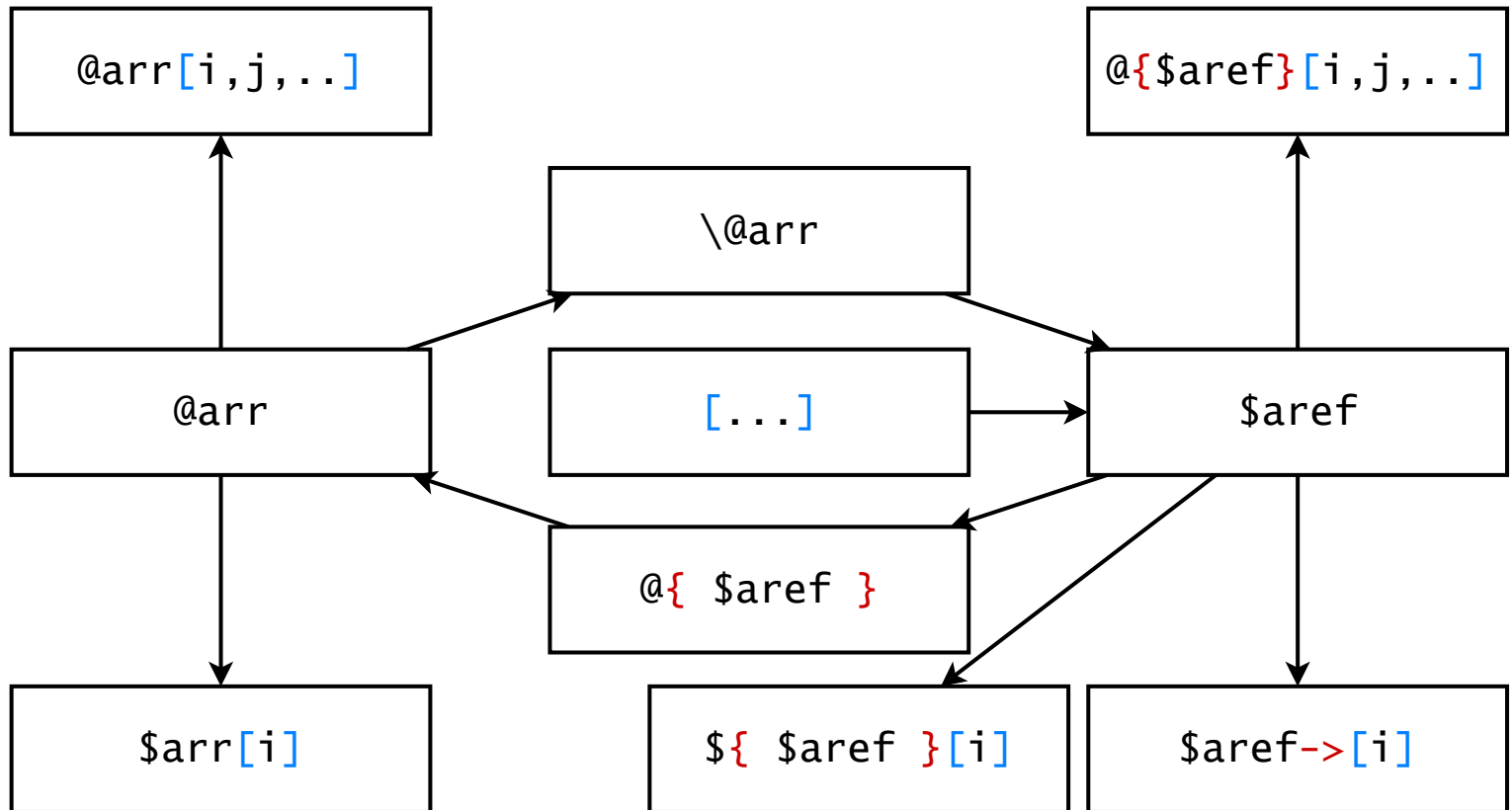
Ссылки

```
$scalarref = \ $scalar;  
$arrayref  = \@array;  
$hashref   = \%hash;  
$coderef   = \&function;  
$globref   = \*FH;  
$refref    = \ $scalarref;  
  
$arrayref  = [ 4, 8, 15, 16 ];  
$hashref   = { one => 1, two => 2 };  
$coderef   = sub { ... };  
  
($a, $b)   = (\"one\", \"two\");  
($a, $b)   = \ (\"one\", \"two\");
```

Переменные: ARRAY & HASH



Переменные: ARRAY



Переменные: ARRAY

```
@simple = qw(1 2 3 bare);  
@array = (4,8,15,16,23,42,@simple);  
@array = (4,8,15,16,23,42,1,2,3,'bare');  
$aref = \@array;  
$aref = [4,8,15,16,23,42,@simple];
```

```
say $array[2];      # 15  
say ${array}[2];  
say ${array}[2];  
say "last i = ", $#array;  
say "last i = ", ${#array};
```

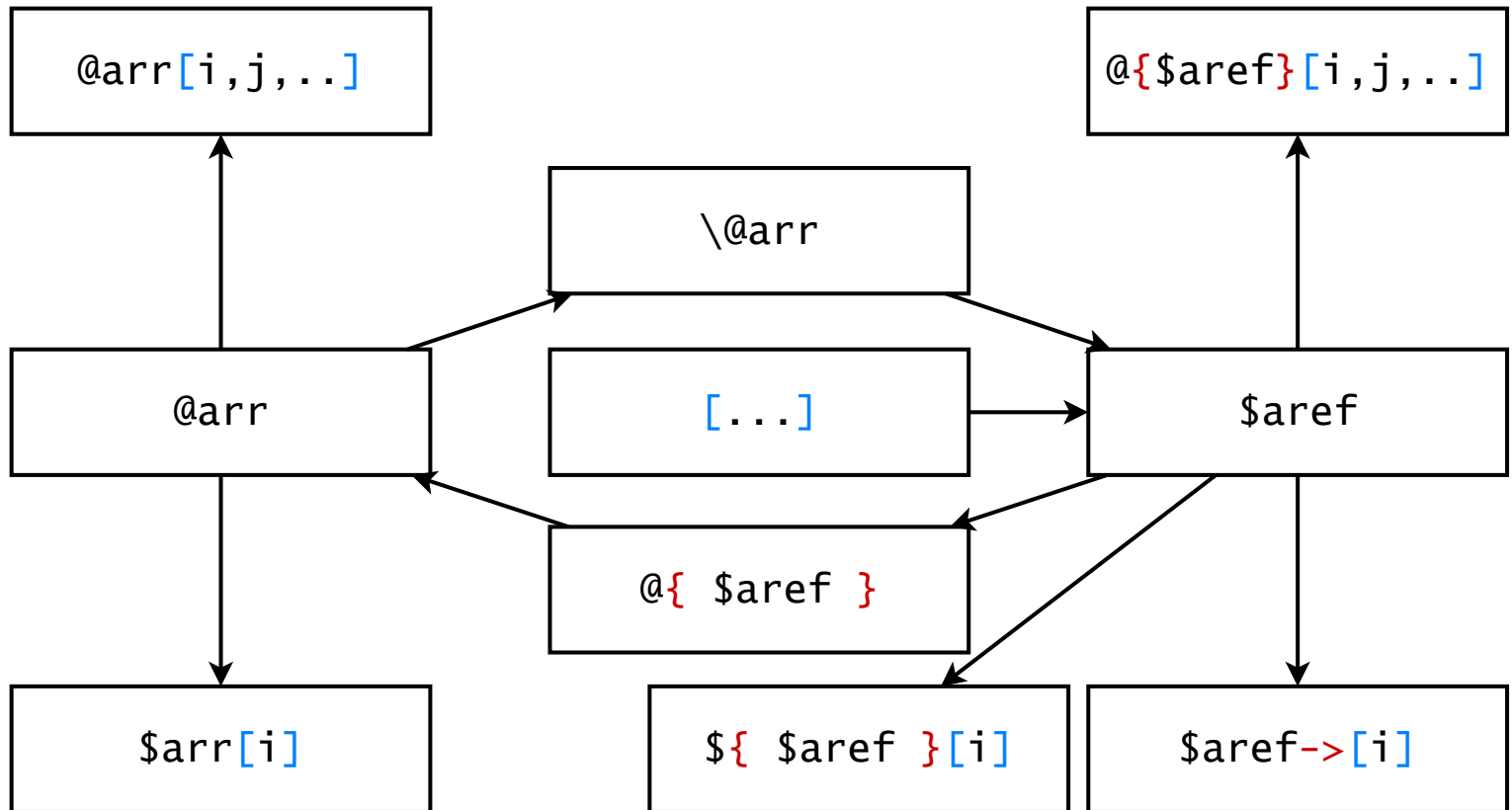
```
say $aref->[2];  
say $$aref[2];  
say ${$aref}[2];  
say "last i = ", $#$aref;  
say "last i = ", $#{aref};  
say "last i = ", ${#{aref}};
```

Переменные: ARRAY

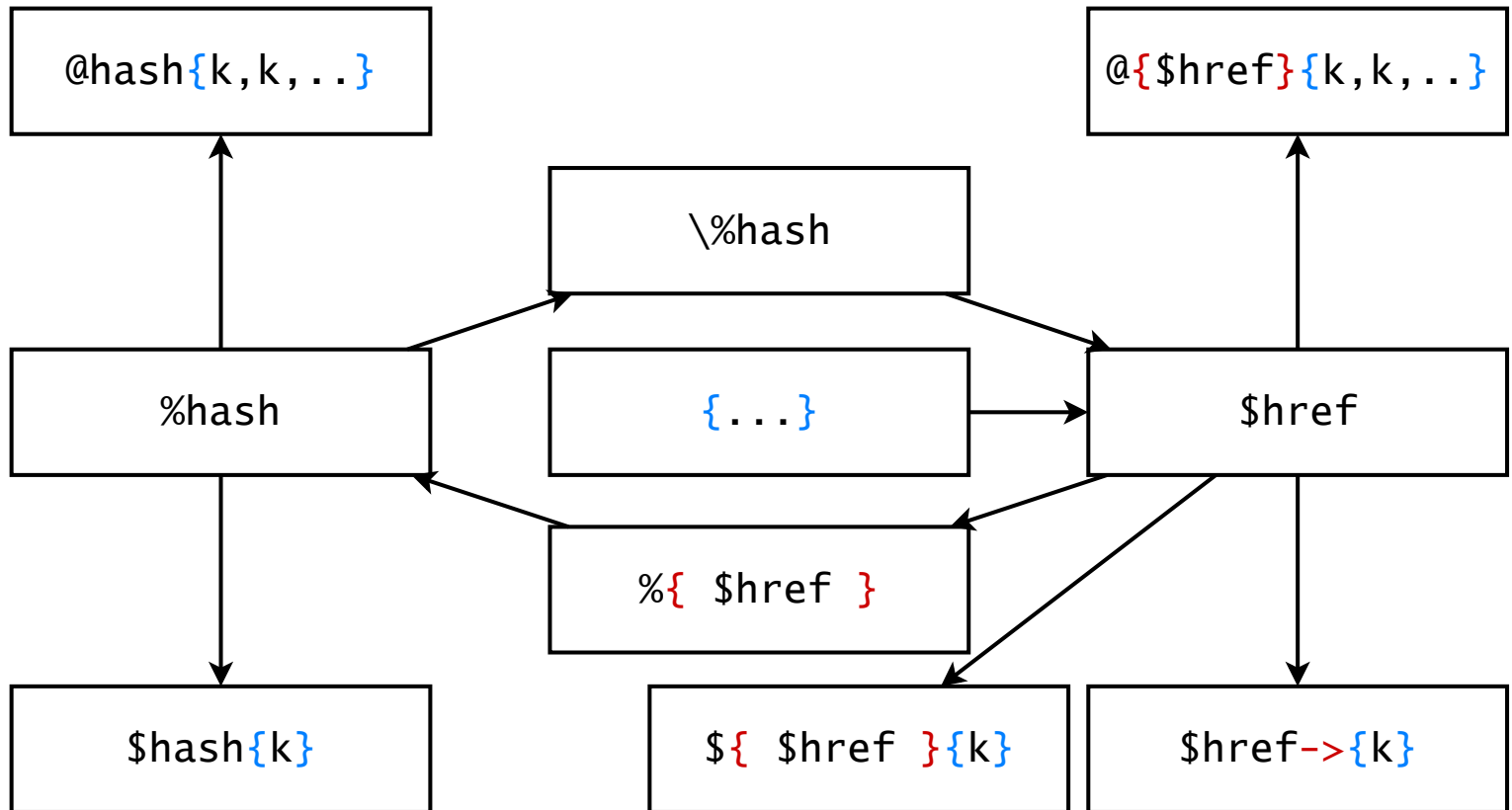
Срезы

```
@simple = qw(1 2 3 bare);  
@array = (4,8,15,16,23,42,@simple);  
$aref = \@array;  
  
say join ", ", @array[0,2,4]; # 4,15,23  
say join ", ", @{array}[0,2,4]; # 4,15,23  
say join ", ", @{ array[0,2,4] }; # 4,15,23  
  
say join ", ", @$aref[0,2,4]; # 4,15,23  
say join ", ", @{ $aref }[0,2,4]; # 4,15,23  
say join ", ", @{ ${aref} }[0,2,4]; # 4,15,23
```

Переменные: ARRAY



Переменные: HASH



Переменные: HASH

```
%simple = qw(k1 1 k2 2);  
%hash = (key3 => 3, 'key4', "four", %simple);  
$href = \%hash; $key = "key3";  
  
say $hash{key3};  
say $hash{"key3"};  
say $hash{ $key };  
say ${hash}{key3};  
say ${ hash{key3} };  
say ${ hash{$key} };  
  
say $href->{key3};  
say $href->{"key3"};  
say $href->{$key};  
say ${href}->{key3};  
say $$ {href}{key3};  
say ${ $href}{key3};  
say ${ ${href} }{key3};
```

Переменные: HASH

```
%simple = qw(k1 1 k2 2);
%hash = (key3 => 3, 'key4', "four", %simple);
$href = \%hash; $key = "key3";

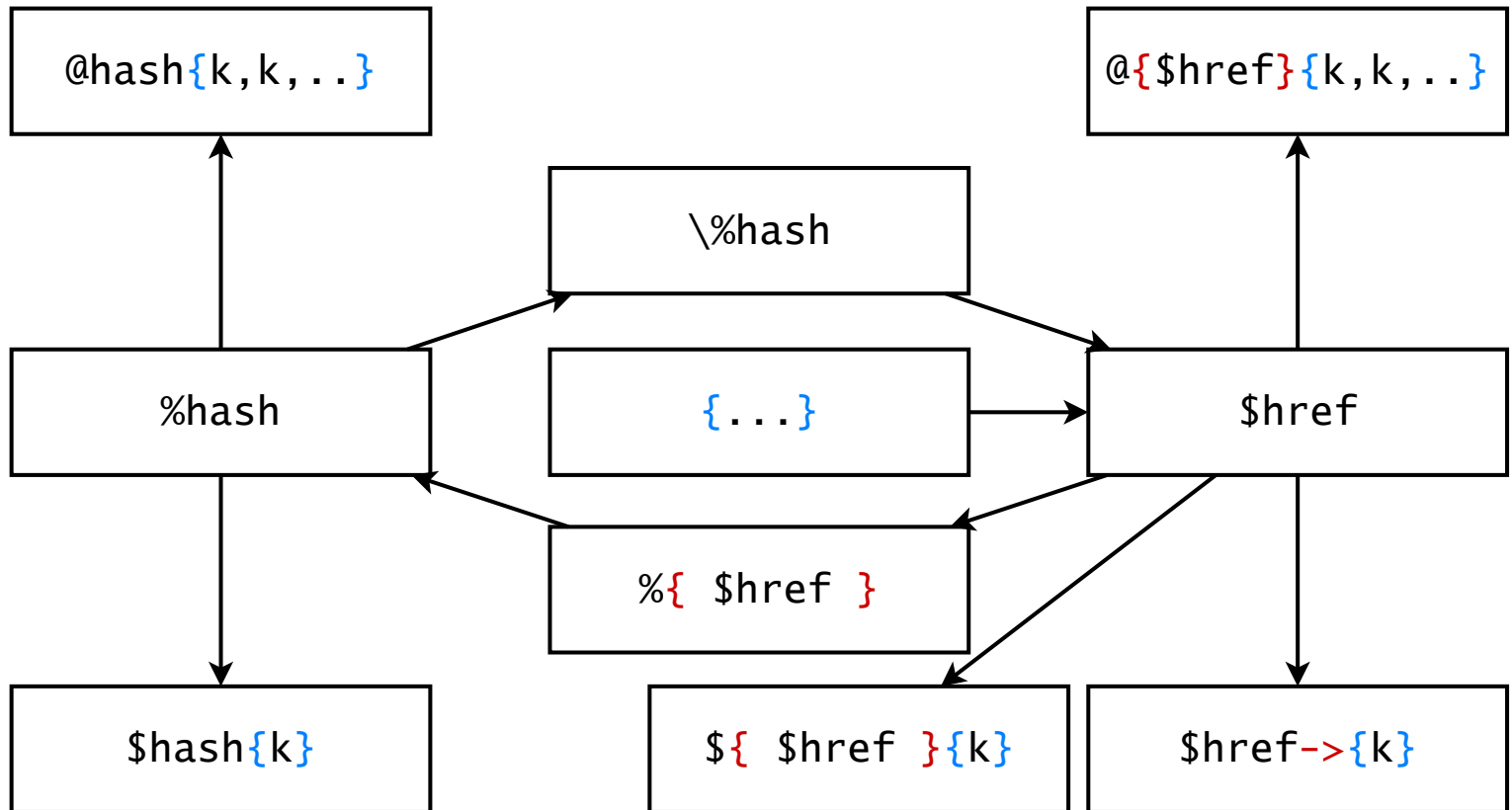
say join ", ", %simple; # k2,2,k1,1
say join ", ", keys %hash; # k2,key3,k1,key4
say join ", ", values %$href; # 2,3,1,four

say join ", ", @hash{ "k1", $key }; # 1,3
say join ", ", @{hash}{ "k1", $key }; # 1,3
say join ", ", @{ hash{ "k1", $key } }; # 1,3

say join ", ", @{$href}{ "k1", "key3" }; # 1,3

$hash{key5} = "five";
$one = delete $href->{k1}; say $one; # 1
say $hash{k2} if exists $hash{k2}; # 2
```

Переменные: HASH



Ссылки - использование

```
$var = 7;
%hash = (
  s => "string",
  a => [ qw(some elements) ],
  h => {
    nested => "value",
    "key\0" => [ 1,2,$var ],
  },
  f => sub { say "ok:@_"; },
);

say $hash{s}; # string
say $hash{a}->[1]; # elements
say $hash{h}->{"key\0"}->[2]; # 7
say $hash{h}{"key\0"}[2]; # 7
$hash{f}->(3); # ok:3
&{ $hash{f} }(3); # ok:3
```


Ссылки - использование

```
$var = 7;
$href = {
  s => "string",
  a => [ qw(some elements) ],
  h => {
    nested => "value",
    "key\0" => [ 1,2,$var ],
  },
  f => sub { say "ok:@_"; },
};

say $href->{s}; # string
say $href->{a}->[1]; # elements
say $href->{h}->{"key\0"}->[2]; # 7
say $href->{h>{"key\0"}[2]; # 7
say $href->{f}->(3); # ok:3
&{ $href->{f} }(3); # ok:3
```

Ссылки - использование

```
$, = ", "; # $OUTPUT_FIELD_SEPARATOR  
@array = (1,2,3);  
say @array; # 1, 2, 3
```

```
@array = [1,2,3];  
say @array; # ARRAY(0x7fcd02821d38)
```

```
%hash = (key => "value");  
say %hash; # key, value
```

```
%hash = {key => "value"};  
say %hash; # HASH(0x7fbbd90052f0),
```

```
%hash = ( key1 => (1,2), key2 => (3,4) );  
say $hash{key1}; # 1  
say $hash{key2}; # undef  
say $hash{2};    # key2  
%hash = ( key1 => 1, 2 => 'key2', 3 => 4 );
```

Ссылки - использование

```
$href = {  
    s => "string",  
};  
  
$href->{none}{key} = "exists";  
say $href->{none};          # HASH(0x7fea...)  
say $href->{none}{key};     # exists  
  
$href->{ary}[7] = "seven";  
say $href->{ary};           # ARRAY(0x7f9...)  
say $href->{ary}[7];        # seven  
say ${ $href->{ary} };      # 7
```

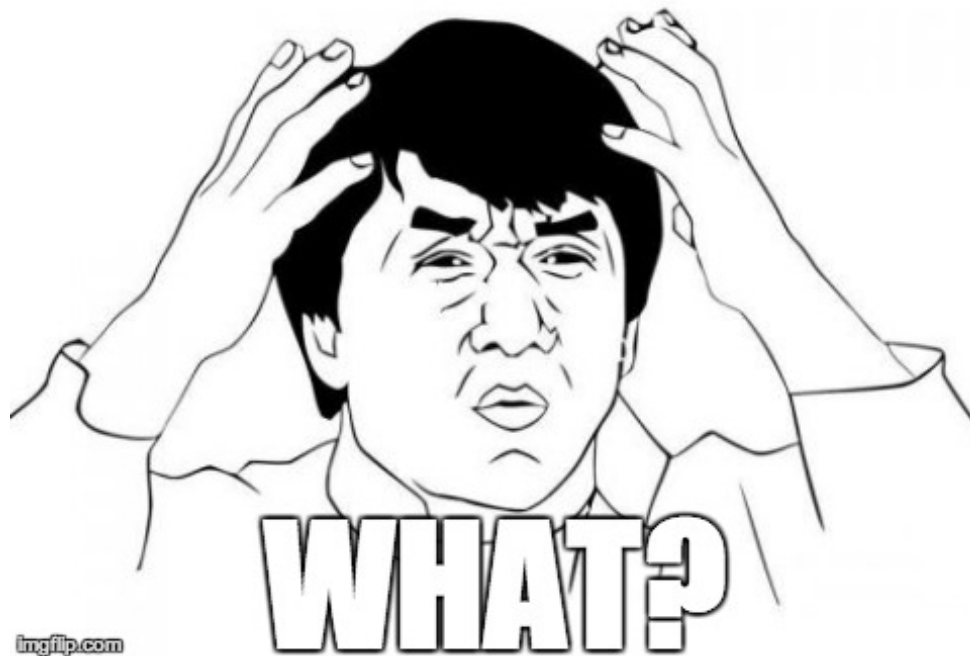
Ссылки - использование

```
$href = {  
    s => "string",  
};  
  
$href->{none}{key} = "exists";  
say $href->{none};          # HASH(0x7fea...)   
say $href->{none}{key};    # exists  
  
$href->{ary}[7] = "seven";  
say $href->{ary};          # ARRAY(0x7f9...)   
say $href->{ary}[7];      # seven  
say ${ $href->{ary} };    # 7  
  
$href->{s}{error} = "what?";  
say $href->{s}{error};    # what?  
say $string{error};      # what?
```

Ссылки - использование

```
$href = {  
  s => "string",  
};  
  
$href->{none}{key} = "exists";  
say $href->{none};          # HASH(0x7fea...)   
say $href->{none}{key};    # exists  
  
$href->{ary}[7] = "seven";  
say $href->{ary};          # ARRAY(0x7f9...)   
say $href->{ary}[7];      # seven  
say ${ $href->{ary} };    # 7  
  
$href->{s}{error} = "what?";  
say $href->{s}{error};    # what?  
say $string{error};      # what?
```

Ссылки - использование



Символические ссылки

Переменная, чьё значение является именем другой переменной

```
$name = "var";  
$$name = 1;      # устанавливает $var в 1  
${$name} = 2;    # устанавливает $var в 2  
@$name = (3,4);  # устанавливает @var в (3,4)  
  
$name->{key} = 7; # создаёт %var и  
                # устанавливает $var{key}=7  
  
$name->();        # вызывает функцию var
```

Символические ссылки

`use strict 'refs'` запрещает их использование

```
use strict 'refs';
```

```
${ bareword }      # ≡ $bareword; # ok
```

```
${ "bareword" }; # not ok
```

```
$hash{ "key1" }{ "key2" }{ "key3" }; # ok
```

```
$hash{ key1 }{ key2 }{ key3 }; # also ok
```

```
$hash{shift}; # ok for keyword, no call
```

```
$hash{ +shift }; # call is done
```

```
$hash{ shift() }; # or so
```


Интерполяция

В строках интерполируются `$..` и `@..`

```
$var = "one";
@ary = (3, "four", 5);
%hash = (k => "v", x => "y");
say "new $var";           # new one
say 'new $var';           # new $var

$" = ' '; # $LIST_SEPARATOR
say "new @ary";           # new 3;four;5
say 'new @ary';           # new @ary
say "1st: $ary[0]";       # 1st: 3
say "<@ary[1,2]>";         # <four;5>
say "<${ ary[1] }>";      # <four>
say "<${hash{x}}>";      # <y>
```

Интерполяция

Инлайновое исполнение:

dereference + reference constructor

```
$var = 100;
say "1+2 = @{[ 1+2 ]}"; # 1+2 = 3
say "\$var/=10 = @{[do{ $var/=10; $var }]}";
# $var/=10 = 10

say "1+2 = ${\ ( 1+2 )}";
say "1+2 = ${\ do{ 1+2 }}";

say "1+2 = ${\ {key=> 1+2 } }{key}";
say "\$var = ${\ {key=> do{ $var } } }{key}";

say "Now: ${\ scalar localtime}";
# Now: Wed Sep 30 19:25:48 2015
```

Содержание

- **Базовый синтаксис**
 - Условия, циклы
 - Управляющие функции
 - Постфиксная нотация
- **Переменные**
 - Основные типы
 - Ссылки
 - Интерполяция
- **Функции**
 - Декларация, аргументы
 - Контекст
 - Прототипы
 - Встроенные функции
 - `grep`, `map`, `sort`
 - `eval`
- **Операторы**
 - Порядок исполнения
 - Особенности операторов
 - Числа и строки

Функции

```
sub NAME;  
sub NAME(PROTO);  
  
sub NAME BLOCK  
sub NAME(PROTO) BLOCK  
  
$sub = sub BLOCK;  
$sub = sub (PROTO) BLOCK;
```

Функции

- Объявление

```
sub mysub;  
...  
sub mysub {  
    @_;          # <- args here  
    my $a = shift; # one arg  
    my ($a,$b) = @_; # 2 args  
    my %h = @_;    # kor k/v  
    say "my arg: ", $_[0];  
  
    return unless defined wantarray;  
    return (1,2,3) if wantarray; # return list  
    1; # implicit ret, last statement  
}
```

Функции

- **Контекст**

```
# scalar context
my $var = mysub(1, 2, $var);
say 10 + mysub();

# list context
@a = mysub();
($x,$y) = mysub();

#void context
mysub();
```

Функции

- **Вызов**

```
mysub( ... );  
mysub ... ;  
&mysub(   );  
&mysub;      # ≡ &mysub( @_ );  
  
$sub->( ... );  
&$sub( ... );  
&$sub;      # ≡ &$sub( @_ );
```

Функции

- Прототипы

```
$ # scalar
@ # list
% # list
* # filehandle
& # special codeblock
; # optional separator

_ # scalar or $_
+ # hash or array (or ref to)
\ # force type
\[ %@ ] # ex: real hash or array

sub vararg($$;$); # 2 req, 1 opt
sub vararg($$;@); # 2 req, 0..* opt
sub noarg(); # no arguments at all
```


Функции

- Прототипы

```
sub check (&@) {  
    my ($code, @args) = @_;  
    for (@args) {  
        $code->($_);  
    }  
}  
  
check {  
    if( $_[0] > 10 ) {  
        die "$_[0] is too big";  
    }  
} 1, 2, 3, 12;
```

Встроенные функции

- `grep`, `map`, `sort`

```
@nonempty = grep { length $_; } @strings;
$count    = grep { length $_; } @strings;
@nonempty = grep length($_), @strings;

@odd  = grep {      $_ % 2 } 1..100;
@even = grep { not $_ % 2 } 1..100;

%uniq = ();
@unique = grep { !$uniq{$_}++ } @with_dups;}

@a = 1..55;
@b = 45..100;
%chk; @chk{@a} = ();
@merge = grep { exists $chk{$_} } @b;
```

Встроенные функции

- `grep`, `map`, `sort`

```
@squares = map { $_**2 } 1..5; # 1,4,9,16,25

say map chr($_), 32..127;

@nums = 1..100;
@sqr = map {
    if( int(sqrt($_)) == sqrt($_) ) {
        $_
    } else { () }
} @nums;

my @reduced =
    map $_->[1],
    grep { int($_->[1]) == $_->[1] }
    map { [$_, sqrt $_] } 1..1000;
```

Встроенные функции

- `grep`, `map`, `sort`

```
@alphabetically = sort @strings;  
@nums = sort { $a <=> $b } @numbers;  
@reverse = sort { $b <=> $a } @numbers;  
@ci = sort { fc($a) cmp fc($b) } @strings;  
  
sub smart { $a <=> $b || fc($a) cmp fc($b) }  
@sorted = sort smart @strings;  
  
my @byval = sort { $h{$a} cmp $h{$b} } keys %h;
```

Встроенные функции

- eval, die, warn

```
eval "syntax:invalid";  
warn $@ if $@;
```

```
eval { $a/$b; };  
warn $@ if $@;
```

```
eval { die "Not root" if $<; };  
warn $@ if $@;
```

```
eval {          # try  
    ...;  
1} or do {      # catch  
    warn "Error: $@";  
};
```

Встроенные функции

- chop, chomp

```
$/ = "\r\n";  
$a = $b = "test\r\n";  
chop($a), chop($a), chop($a); # \n, \r, t  
say $a;  
chomp($b), chomp($b) # \r\n, '';  
say $b;
```

- index, rindex, substr, length

```
#           ↓—————index($_, " ") # 4  
$_ = "some average string\n";  
#           └───┬────────↑—————rindex($_, " ") # 12  
#           substr($_, 3, 5) = "e ave"
```

Встроенные функции

- `lc`, `lcfirst`, `uc`, `ucfirst`, `fc`

```
$big = "WORD"; $small = "word";  
say lc $big;          # word "\L"  
say lcfirst $big;     # wORD "\l"  
say uc $small;        # WORD "\U"  
say ucfirst $small;   # Word "\u"  
  
say "equal" if  
    fc $big eq fc $small; # v5.16+  
  
say "\u\LnAmE\E"; # Name
```

Встроенные функции

- `chr`, `ord`, `hex`, `oct`

```
use utf8; use open qw(:utf8 :std);
say chr(80);           # P
say ord("P");          # 80
say ord(chr(-1));      # 65533, BOM \x{fffd}
say ord("±");          # 177
say chr(177);          # ±
say chr(9786);         # ☺
say ord "ë";           # 1105
say hex "dead_beaf";   # 3735928495
say hex "0xDEAD";      # 57005
say oct "04751";       # 2537
```


Встроенные функции

- `reverse`, `sprintf`

```
# Why
say reverse 'dog';
# prints dog,
# but
say ucfirst reverse 'dog';
# prints God?
```

```
$a = sprintf"%c %s %d %u\n%o %x %e %f %g",
    9786, "str", -42, -1, 2537,
    57005, 1/9, 1/3, .6626E-33;
say $a;
# ☺ str -42 18446744073709551615
# 4751 dead 1.111111e-01 0.333333 6.626e-34
```

Встроенные функции

- `abs`, `int`, `srand`, `rand`, `log`, `exp`, `sin`, `cos`, `atan2`, `sqrt`

```
while (<>) {  
    say int( log ( abs $_ ) / log 10 );  
}  
  
printf "%g\n", log ( 1e6 ) / log( 10 ) - 6 ;  
# -8.88178e-16
```

Встроенные функции

- `each`, `keys`, `values`

```
%h = map { $_ => -$_ } 1..3;  
@a = keys %h; @b = values %h;  
while (my ($k, $v) = each @a)  
    { say "$k: $v ($b[$k])"; }  
while (my ($k, $v) = each %h)  
    { say "$k: $v"; }
```

```
0: 1 (-1)  
1: 3 (-3)  
2: 2 (-2)  
1: -1  
3: -3  
2: -2
```

Встроенные функции

- push, pop, shift, unshift, splice

```
push(@a, $x, $y)      splice(@a, @a, 0, $x, $y)
pop(@a)               splice(@a, -1)
shift(@a)             splice(@a, 0, 1)
unshift(@a, $x, $y)   splice(@a, 0, 0, $x, $y)
$a[$i] = $y           splice(@a, $i, 1, $y)
```

```
@a = ( 1, 2, 3, 4, 5, 6, 7 );
#
#
#
splice( @a, 1, 3, ( 8, 9 ) );
say @a;
# 1, 8, 9, 5, 6, 7
```

The diagram illustrates the splice operation. It shows the original array @a = (1, 2, 3, 4, 5, 6, 7). The splice function is called with @a, 1, 3, and a replacement array (8, 9). The diagram shows the elements 2, 3, and 4 being removed and replaced by 8 and 9. The final array is (1, 8, 9, 5, 6, 7).

Встроенные функции

- `gmtime`, `localtime`, `time`, `strftime`

```
say time;                # 1443632440
say ~~localtime;         # Wed Sep 30 20:00:40 2015
say ~~localtime 0;       # Thu Jan  1 03:00:00 1970
say ~~gmtime 0;          # Thu Jan  1 00:00:00 1970

($s,$m,$h,$D,$M,$Y,$Wd,$Yd,$dst) =
    localtime( time+86400 );
printf "%04u-%02u-%02uT%02u:%02u:%02u",
    $Y+1900, $M+1, $D, $h, $m, $s;
printf "Day no: %u, Weekday: %u", $Yd, $Wd;

# 2015-10-01T20:00:40
# Day no: 273, Weekday: 4

use POSIX 'strftime';
say strftime "%r", localtime(); # 08:00:40 PM
```

Встроенные функции

- `caller`, `goto`

```
sub test1 {  
  my $i=0;  
  while ( ($pk, $f, $l,$s) = caller($i++)) {  
    say "$i. from $f:$l ($s)";  
  }  
}  
sub test2 {  
  test1()  
};  
sub test3 {  
  test2();  
}  
sub test4 { goto &test2; }  
test3();  
test4();
```

Содержание

- **Базовый синтаксис**
 - Условия, циклы
 - Управляющие функции
 - Постфиксная нотация
- **Переменные**
 - Основные типы
 - Ссылки
 - Интерполяция
- **Функции**
 - Декларация, аргументы
 - Контекст
 - Прототипы
 - Встроенные функции
 - `grep`, `map`, `sort`
 - `eval`
- **Операторы**
 - Порядок исполнения
 - Особенные операторы
 - Числа и строки

Операторы

Оператор определяет тип операнда

Ассоциативность и приоритет арифметических операторов соответствует тому, как это принято в математике

Приоритеты операторов

| ассоциативность | оператор |
|-----------------|--|
| left | TERM и LIST (leftward) |
| left | -> |
| n/a | ++, -- |
| right | ** |
| right | ! ~ \, unary +, - |
| left | = ~ ! ~ |
| left | * / % x |
| left | + - . |
| left | << >> |
| n/a | named unary ops # (функции с одним аргументом) |
| n/a | < > <= >= lt gt le ge |
| n/a | == != <=> eq ne cmp ~~ |
| left | & |
| left | ^ |
| left | && |
| left | // |
| n/a | :: ... |
| right | ? : |
| right | = += -= *= etc. |
| left | , => |
| n/a | LIST (rightward) |
| right | not |
| left | and |
| left | or xor |

Операторы TERM, LIST(L)

- Любая переменная (`$variable`)
- Обращение к хэшу или массиву (`$hash{key}` или `$array[$x]`)
- Любая строка (`"string"` или `'str'`), число (`42`, `-1e42`) или quote-like оператор
- Любой вызов функции со скобками `func(...)`
- `do{ ... }, eval{ ... }, sub{ ... }`
- Анонимные конструкторы
`{ ... }` и `[...]`

Операторы TERM, LIST(L)

```
my $v = 5;  
my @a = ( 1, 2, sort 3, 4+$v, 6x2, 7 );
```

- -M0=Deparse, -p

```
(my $v = 5);  
(  
    my @a = (  
        1, 2,  
        sort(  
            3,  
            (4 + $v),  
            (6 x 2),  
            7  
        )  
    )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1, # 1  
    2, # 2  
    sort(  
      3,  
      (  
        4  
        +  
        $v  
      ),  
      (  
        6  
        x  
        2  
      ),  
      7  
    )  
  )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4  
        +  
        $v  
      ),  
      (  
        6  
        x  
        2  
      ),  
      7  
    )  
  )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4        # 4  
        +        # 6  
        $v       # 5  
      ),  
      (  
        6  
        x  
        2  
      ),  
      7  
    )  
  )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4        # 4  
        +        # 6  
        $v       # 5  
      ),  
      (  
        6        # 7  
        x        # 9  
        2        # 8  
      ),  
      7  
    )  
  )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4        # 4  
        +        # 6  
        $v       # 5  
      ),  
      (  
        6        # 7  
        x        # 9  
        2        # 8  
      ),  
      7          # 10  
    )  
  )  
);
```


Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4        # 4  
        +        # 6  
        $v       # 5  
      ),  
      (  
        6        # 7  
        x        # 9  
        2        # 8  
      ),  
      7          # 10  
    )           # 11  
  )  
);
```

Операторы TERM, LIST(L)

```
(  
  my @a = (  
    1,          # 1  
    2,          # 2  
    sort(  
      3,        # 3  
      (  
        4        # 4  
        +        # 6  
        $v       # 5  
      ),  
      (  
        6        # 7  
        x        # 9  
        2        # 8  
      ),  
      7          # 10  
    )          # 11  
  )          # 12  
);
```

Оператор стрелочка ()

суффиксный оператор разыменования
(infix dereference)

```
STMT->{...} # STMT должен вернуть HASHREF
```

```
STMT->[...] # STMT должен вернуть ARRAYREF
```

```
STMT->(...) # STMT должен вернуть CODEREF
```

```
STMT->method(...)
```

```
# STMT должен вернуть объект или класс
```

```
STMT->$var(...)
```

```
# $var должен вернуть имя метода или CODEREF
```

Операторы инкремента

Аналогичны соответствующим в C
(auto-increment and auto-decrement)

```
my $i = 0;  
$x = $i++; # $x = 0; $i = 1;  
$y = ++$i; # $y = 2; $i = 2;
```

- `++$i + $i++` - неопределённое поведение
- `undef` - всегда как число 0

Операторы инкремента

немного "магии"

- Если `$var` строка
- Начинается на `[a-z]` или `[A-Z]`
- Содержит `[a-z]`, `[A-Z]` или `[0-9]`

```
say ++($a = "a");    #    b
say ++($a = "aa");   #    ab
say ++($a = "AA");   #    AB
say ++($a = "Aa1");  #    Aa2
say ++($a = "Aa9");  #    Ab0
say ++($a = "Az9");  #    Ba0
say ++($a = "Zz9");  #    AAa0
say ++($a = "zZ9");  #    aaA0
```

Декремент магическим не является

Унарные операторы - !

логическое отрицание

logical negation

False: 0, "", undef, overloaded obj

True: остальное, по умолчанию 1

```
!0      # 1
!1      # ""
!"      # 1
!undef  # 1
```

Унарные операторы - -

математическое отрицание

arithmetic negation

- специальное поведение на строках

```
-0      # 0
-1      # -1
- -1    # 1
-"123"  # -123
- "-123" # 123
-undef  # 0 or -0
-bare    # "-bare"
-"word"  # "-word"
- "+word" # "-word"
- "-word" # "+word"
- "-0"   # 0 or +0
```

Унарные операторы - ~

битовая инверсия

bitwise negation

```
# numbers
~0      # 0xffff_ffff or 0xffff_ffff_ffff_ffff
0777 & ~027 # 0750 (in oct)

# byte string
~"test"  # "\213\232\214\213"
          # chr(~ord("t")).chr(~ord("e")).
          # chr(~ord("s")).chr(~ord("t"));

# char string
use utf8;
"ë" # ≡ "\x{451}"
~"ë" # "\x{fffffbæ}" 32b
~"ë" # "\x{fffffffffffffbæ}" 64b
```


Унарные операторы - +

унарный плюс

unary "+"

Не имеет эффекта,

используется как разделитель

```
say +( 1 + 2 ) * 3; # 9
```

```
say ( 1 + 2 ) * 3; # 3
```

```
return +{}; # empty anon hash
```

```
map { +{ $_ => -$_ } } @_;
```

Операторы - `=~`, `!~`

Применение регэкспа
`match, binding`

```
# match
$var =~ /regexp/;
$var !~ /shoudn't/;

# replace
$var =~ s/word/bare/g;

# transliterate
$var =~ tr/A-Z/a-z/;

if ($var = ~ /match/) {...} # always true
#           ^
#           '----- beware of space
if ($var = (~(/match/))) {...}
```

Операторы - * / % x

Умножение, деление, остаток, повтор
multiply, divide, modulo, repeat

```
say 9*7; # 63
say 9/7; # 1.28571428571429
say 9%7; # 2

say 9x7; # 99999999
say join ",",(9,10)x3; # 9,10,9,10,9,10
```

Операторы - + - .

Сложение, вычитание, конкатенация
add, subtract, concat

```
say 1+2;           # 3
say "1"+"2";       # 3
say "1z" + "2z";   # 3
say "a" + "b";     # 0

say 1.2;           # 1.2
say 1 . 2;         # 12
say "1"."2";       # 12
say "a"."b";       # ab
```

Операторы - << >>

Сдвиг влево, сдвиг вправо
shift-left, shift-right

Реализовано полностью с использованием C
Поведение аналогично

```
say 20 << 20; # 20971520
say 20 << 40; # 5120 on 32-bit
               # 2199023255520 on 64-bit
use bigint;
say 20 << 80; # 24178516392292583494123520
```

Операторы сравнения

`<, >, <=, >=, lt, gt, le, ge`

relational operators

```
say "a" > "b";      # "", 0 > 0
say "a" < "b";      # "", 0 < 0

say 100 gt 20;      # "", "100" gt "20"
say "100" > "20";  # 1
```

- `<, >, <=, >=` - всегда преобразуют к числу
- `lt, gt, le, ge` - преобразуют к строке

Операторы равенства

`==`, `!=`, `<=>`, `eq`, `ne`, `cmp`

equality operators

```
say 10 == "10";      # 1
say "20" != "10";    # 1
say 1 <=> 2;          # -1
say 1 <=> 1;          # 0
say 2 <=> 1;          # 1
say "a" <=> "b";      # 0
say "a" == "b";      # 1

say 1 eq "1";        # 1
say "0" ne 0;        # ""
say "a" cmp "b";     # -1
say "b" cmp "a";     # 1

say "No NaN" if "NaN" == "NaN";
```

Оператор умного сравнения

`~~`, perl 5.10.1+

smartmatch operator

experimental in 5.18+

```
my @ary = (1,2,undef,7);
say "sparse" if undef ~~ @ary;

given ($num) {
    when ([1,2,3]) { # as $num ~~ [1,2,3]
        say "1..3";
    }
    when ([4..7]) { # as $num ~~ [4..7]
        say "4..7";
    }
}
```


Битовые операторы

&, |, ^

and, or, xor

```
$x = int(rand(2**31-1));  
say $x & ~$x + 1;  
say $x ^ ( $x & ($x - 1));
```

```
$x = $x ^ $y;  
$y = $y ^ $x;  
$x = $x ^ $y;
```

```
say "test" ^ "^^^^"; # *; -*  
say "test" & "^^^^"; # TDRT
```

C-style логические операторы

&&, ||, //

and, or, defined-or

- Выполняются последовательно
- Передают контекст
- Возвращают последнее значение

```
say 1 && "test"; # test
say 0 || "test"; # test
say 1 || die;    # 1    # say( 1 || die );
say 0 && die;    # 0    # say( 0 && die );

$a = $x // $y;
$a = defined $x ? $x : $y;
```

Операторы диапазона

..., ...

range operators

СПИСКОВЫЙ КОНТЕКСТ

```
@a = 1..10;  
  
for ("a".."z") {  
    say $_;  
}  
  
say "A"..."Z";  
  
@b = @a[3..7];
```

Операторы диапазона

..., ...

range operators

скалярный (логический) контекст константный операнд

```
for $. (1..5) { # $. - $INPUT_LINE_NUMBER
    say "$. : ".(2..4);
}
```

```
1 :
2 : 1      # became true ($. == 2)
3 : 2      # stay true, while $. != 4
4 : 3E0    # ret true, $. == 5, became false
5 :
```

Операторы диапазона

..., ...

range operators

скалярный (логический) контекст операнд - выражение

```
for $. (1..5) { print "$.";
  do {
    print "\tL,ret " . ($.==2?"T":"F");
    $. == 2;
  } `..` do {
    print "\tR,ret " . ($.==4?"T":"F");
    $. == 4;
  };
  say " : ".scalar(2..4);
}
```

Операторы диапазона

..., ...

range operators

скалярный (логический) контекст операнд - выражение

```
1  L,ret F :  
2  L,ret T R,ret F : 1  
3  R,ret F : 2  
4  R,ret T : 3E0  
5  L,ret F :
```

Операторы диапазона

..., ...

range operators

скалярный (логический) контекст операнд - выражение

```
for $. (1..5) { print "$.";
  do {
    print "\tL,ret " . ($.==2?"T":"F");
    $. == 2;
  } `...` do {
    print "\tR,ret " . ($.==4?"T":"F");
    $. == 4;
  };
  say " : ".scalar(2...4);
}
```

Операторы диапазона

..., ...

range operators

скалярный (логический) контекст операнд - выражение

```
1  L,ret F :  
2  L,ret T : 1  
3  R,ret F : 2  
4  R,ret T : 3E0  
5  L,ret F :
```


Тернарный оператор

?:

ternary operator, as in C

```
$a = $ok ? $b : $c;
```

```
@a = $ok ? @b : @c;
```

```
($a_or_b ? $a : $b) = $c;
```

Оператор присваивания

=

assignment operator, as in C

- += -=
- *= /= %= **=
- &= |= ^= <<= >>=
- &&= ||= // =

Оператор запятая

, запятая, => жирная запятая

comma, fat comma

```
$a = do { say "one"; 3 }, do { say "two"; 7};  
# $a = 7. 3 thrown away
```

```
@list = (bareword => STMT);
```

```
# forces "" on left
```

```
@list = ("bareword", STMT);
```

```
use constant CONST => "some";
```

```
%hash = ( CONST      => "val"); # "CONST"
```

```
%hash = (+CONST      => "val"); # "CONST"
```

```
%hash = ( CONST()    => "val"); # "some"
```

```
%hash = (&CONST       => "val"); # "some"
```

Низкоприоритетные логические операторы

and, or, xor, not

- операторы с низжайшим приоритетом

```
open $file, "<", "0" || die "Can't";
open $file, "<", ( "0" || die "Can't" );
open $file, "<", "0" or die "Can't";
open ( $file, "<", "0" ) or die "Can't";

do_one() and do_two() or do_another();

@info = stat($file) || say "error";
#                               ^-cast scalar context
@info = stat($file) or say "error";
#                               ^-keep list context
```

Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- q - строка без интерполяции

```
say 'string';  
say q{string};  
say q/string/;  
say q;string;;  
say q{str{i}ng}; # balanced  
say q qtestq;  
say q{str{ing}}; # not ok, unbalanced }
```

Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- qq - строка с интерполяцией

```
say "perl $^V";  
say qq{perl $^V};  
say qq/perl $^V/;  
say qq;perl $^V;;  
say qq{perl $^V};
```

Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- `qw` - генератор списка (без интерполяции)

```
$, = ' , ' ;
```

```
say qw(a b c);
```

```
# say split / /, 'a b c';
```

```
for (qw(/usr /var)) {  
    say stat $_;  
}
```

Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- qx - внешняя команда
 - с интерполяцией
 - qx'...' - без интерполяции

```
say qx{uname -a};
```

```
say qx'echo $HOME';
```


Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- `qr` - сборка регкспа
- `/.../`, `m` - сопоставление (match)
- `s` - поиск/замена (replace)
- `y`, `tr` - транслитерация

```
$re = qr/\d+/;  
if ( $a =~ m[test${re}] ) { ... }  
$b =~ s{search}[replace];  
y/A-Z/a-z/; # on $_
```

Оператор кавычки

q qq qw qx qr s y tr

quote-like operators

- Here-doc

```
say <<EOD;  
Content of document  
EOD
```

```
say(<<'THIS', "but", <<THAT);  
No $interpolation  
THIS  
For $ENV{HOME}  
THAT
```

Содержание

- **Базовый синтаксис**
 - Условия, циклы
 - Управляющие функции
 - Постфиксная нотация
- **Переменные**
 - Основные типы
 - Ссылки
 - Интерполяция
- **Функции**
 - Декларация, аргументы
 - Контекст
 - Прототипы
 - Встроенные функции
 - `grep`, `map`, `sort`
 - `eval`
- **Операторы**
 - Порядок исполнения
 - Особенности операторов
 - Числа и строки

Список документации

- [perlsyn](#)
- [perldata](#)
- [perlref](#)
- [perllo](#)
- [perlsub](#)
- [perlfunc](#)
- [perlop](#)
- [perlglossary](#)

Домашнее задание

- Калькулятор

1. Синтаксический разбор и выполнение арифметики
2. Преобразование в обратную польскую нотацию

| | |
|----------|-------------------------------------|
| () | - приоритет 0 |
| - | - унарный минус, приоритет 1 |
| ** или ^ | - возведение в степень, приоритет 2 |
| * / | - умножение, приоритет 3 |
| + - | - сложение, приоритет 4 |

```
# входное выражение:
- 16 + 2 * 0.3e+2 - .5 ^ ( 2 - 3 )
# с расставленными скобками
( -16 ) + ( 2 * ( 30.0 ) ) - ( 0.5 ^ ( 2 - 3 ) )
# значение на выходе
42
# обратная польская нотация
-16 2 30 * + 0.5 2 3 - ^ -
```

__END__

Bonus tracks

Запись однострочников в файл

sample.pl

```
#!/usr/bin/env perl -pl00730012
```

```
perl -M0=Deparse sample.pl
```

```
BEGIN { $/ = "\n"; $\ = ";"; }  
LINE: while (defined($_ = <ARGV>)) {  
    chomp $_;  
}  
continue {  
    die "-p destination: $_\n" unless print $_;  
}  
sample.pl syntax OK
```

Постфиксные циклы

- Добавка внутреннего блока

```
do {{  
    next if $cond1;  
    redo if $cond2;  
    ...  
}} while ( EXPR );
```

- Работает next
- Работает redo
- Не работает last

Постфиксные циклы

- Добавка внешнего блока

```
LOOP: {  
    do {  
        next if $cond1;  
        redo if $cond2;  
        ...  
    } while ( EXPR );  
}
```

- Работает `last`
- Не работает `redo` *
- Не работает `next` *

* А точнее, работает не так, как ожидалось

Постфиксные циклы

- Добавка внешнего и внутреннего блока

```
LOOP: {  
    do {{  
        next if $cond1;  
        redo if $cond2;  
        last LOOP if $cond3;  
        ...  
    }} while ( EXPR );  
}
```

- Работает `last` (по внешней метке)
- Работает `redo`
- Работает `next`

* Главное не ошибиться с метками

Особенности perl 5.20: постфиксное разыменование

```
use feature 'postderef';  
no warnings 'experimental::postderef';
```

```
$sref->$*;    # same as ${ $sref }  
$aref->@*;    # same as @{ $aref }  
$aref->$#*;   # same as $#{ $aref }  
$href->%*;    # same as %{ $href }  
$cref->&*;    # same as &{ $cref }  
$gref->**;    # same as *{ $gref }
```

Особенности perl 5.20: срезы ключ/значение

```
%hash = (  
    key1 => "value1",  
    key2 => "value2",  
    key3 => "value3",  
    key4 => "value4",  
);  
#%sub = (  
#    key1 => $hash{key1},  
#    key3 => $hash{key3},  
#);  
%sub = %hash{"key1", "key3"};  
      ^      ^              ^  
      |      +-----+-----+----- на хэше  
      +-----+-----+----- хэш-срез
```

Особенности perl 5.20: срез ключ/значение на массиве

```
@array = (  
    "value1",  
    "value2",  
    "value3",  
    "value4",  
);  
#%sub = (  
#     1 => $array[1],  
#     3 => $array[3],  
# );  
%sub = %array[ 1, 3 ];  
           ^      ^      ^  
           |      +-----+---- на массиве  
           +-----+----- хэш-срез
```

Особенности perl 5.20: постфиксный срез

```
($a,$b) = $aref->@[ 1,3 ];  
($a,$b) = @{ $aref }[ 1,3 ];
```

```
($a,$b) = $href->@{ "key1", "key2" };  
($a,$b) = @{ $href }{ "key1","key2" };
```

```
%sub = $aref->%[ 1,3 ];  
%sub = %{ $aref }[1,3];  
%sub = (1 => $aref->[1], 3 => $aref->[3]);
```

```
%sub = $href->%{ "k1","k3" };  
%sub = %{ $href }["k1","k3"];  
%sub = (k1 => $href->{k1},  
        k3 => $href->{k3});
```



```
use feature 'signatures';
sub foo ($x, $y) {
    return $x**2+$y;
}

sub foo {
    die "Too many arguments for subroutine"
        unless @_ <= 2;
    die "Too few arguments for subroutine"
        unless @_ >= 2;
    my $x = $_[0];
    my $y = $_[1];
    return $x**2 + $y;
}
```

Именованные унарные операторы

функции, имеющие строго 1 аргумент

named unary operators

```
chdir $foo      || die; # (chdir $foo) || die
chdir($foo)     || die; # (chdir $foo) || die
chdir ($foo)    || die; # (chdir $foo) || die
chdir +($foo)   || die; # (chdir $foo) || die

rand 10 * 20;    # rand (10 * 20)
rand(10) * 20;  # (rand 10) * 20
rand (10) * 20; # (rand 10) * 20
rand +(10) * 20; # rand (10 * 20)

-e($file).".ext" # -e( ($file).".ext" )
```

