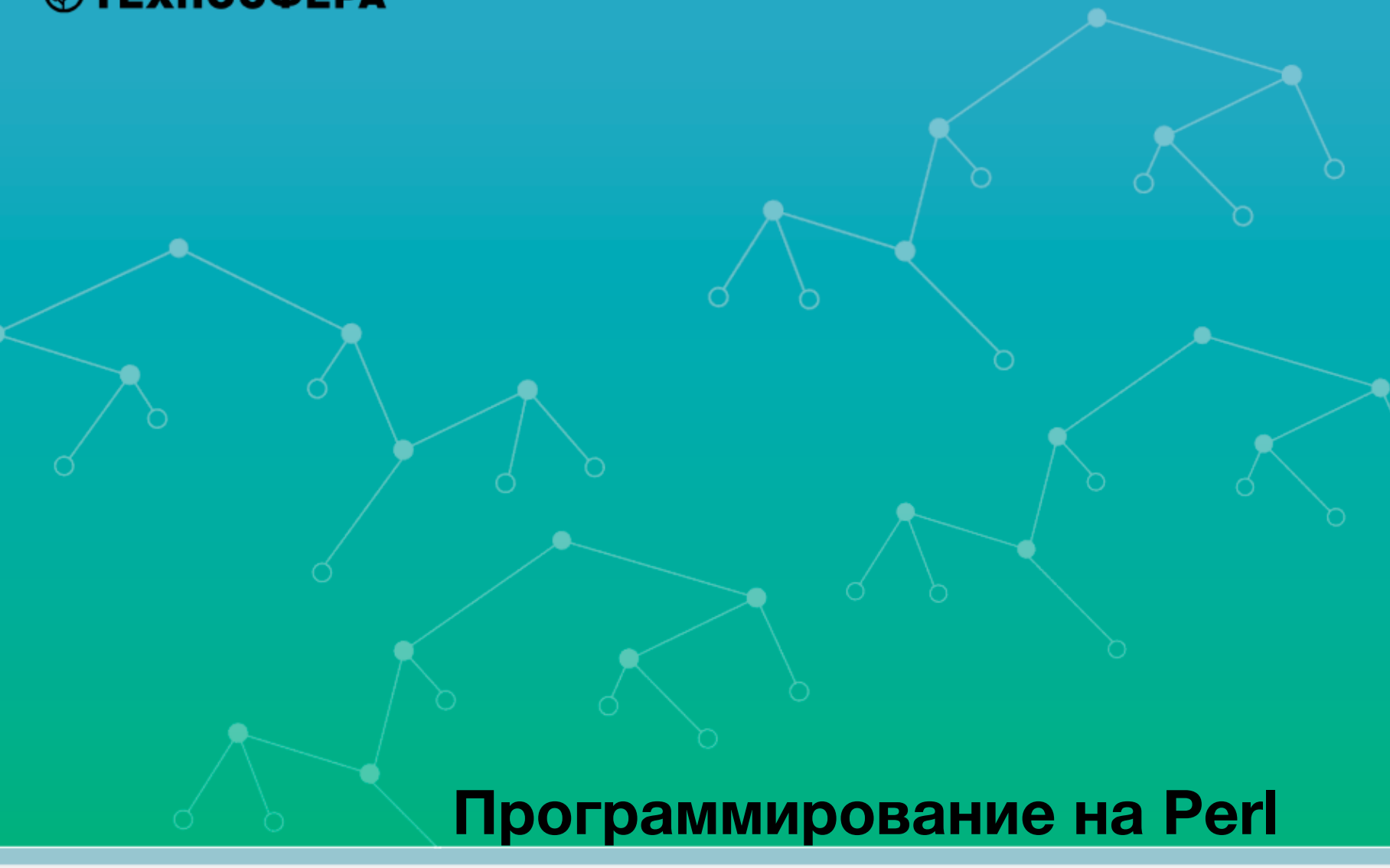
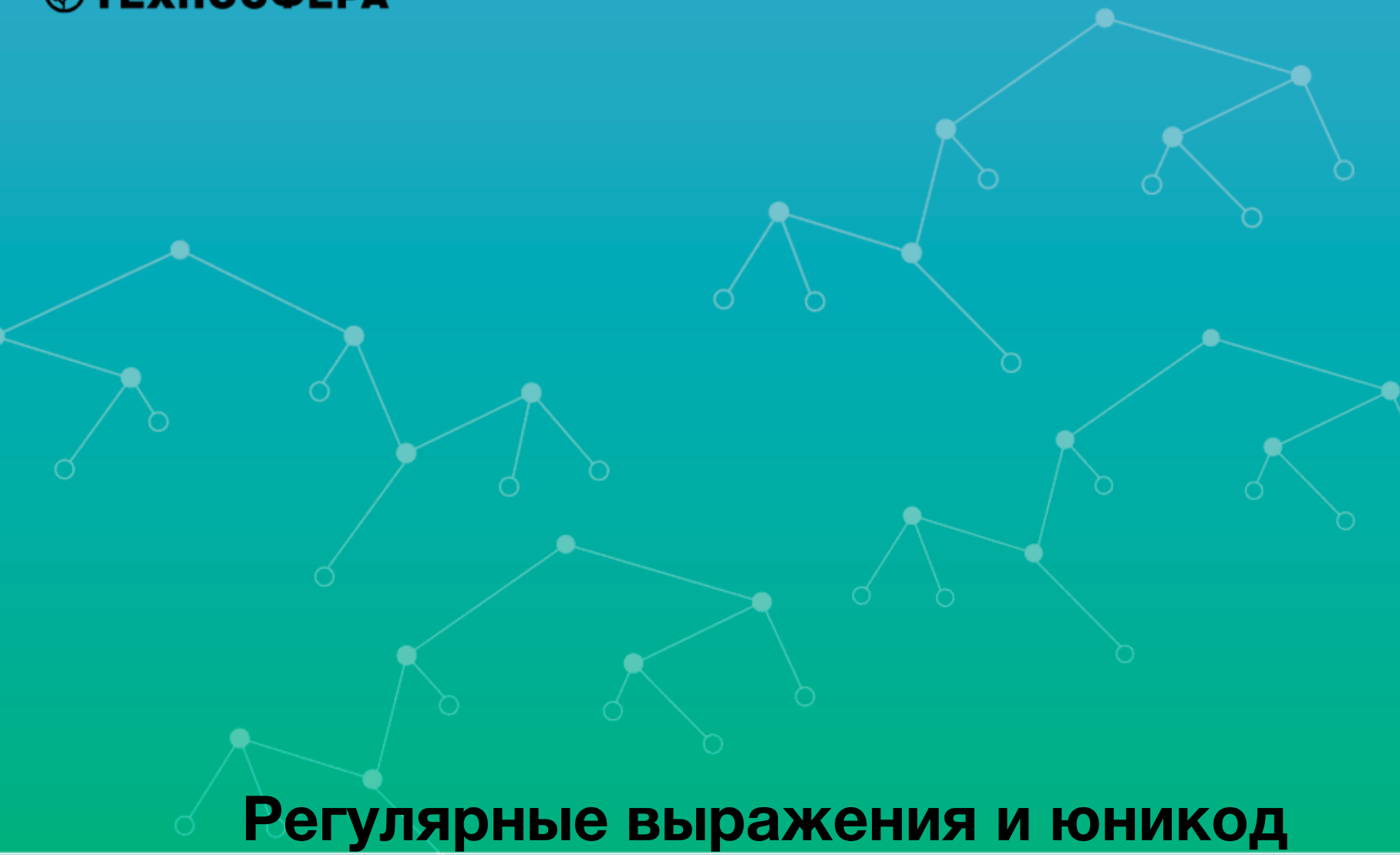
ТЕХНОСФЕРА



Программирование на Perl

ТЕХНОСФЕРА



Регулярные выражения и юникод

Содержание

Unicode

Стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков

Даже Клингонского)

А также разнообразных специальных символов


unicode-table.com/ru/
0 1 2 3 4 5 6 7 8 9 A B C D E F
03F0 ϰ ϱ ϲ ϳ ϴ ϵ ϶ Ϸ ϸ Ϲ Ϻ ϻ ϼ Ͻ Ͼ Ͽ
0400 Ѐ Ё Ђ Ѓ Є Ѕ І Ї Ј Љ Њ Ћ Ќ Ѝ Ў Џ
0410 А Б В Г Д Е Ж З И Й К Л М Н О П
0420 Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я
0430 а б в г д е ж з и й к л м н о п
0440 р с т у ф х ц ч ш щ ъ ы ь э ю я
0450 ѐ ё ђ ѓ є ѕ і ї ј љ њ ћ ќ ѝ ў џ
0460 Ѡ ѡ Ѣ ѣ Ѥ ѥ Ѧ ѧ Ѩ ѩ Ѫ ѫ Ѭ ѭ Ѯ ѯ
0470 Ѱ ѱ Ѳ ѳ Ѵ ѵ Ѷ ѷ Ѹ ѹ Ѻ ѻ Ѽ ѽ Ѿ ѿ
0480 Ҁ ҁ ҂ ҃ ҄ ҅ ҆ ҇ ҈ ҉ Ҋ ҋ Ҍ ҍ Ҏ ҏ
0490 Ґ ґ Ғ ғ Ҕ ҕ Җ ж Ҙ ҙ Қ қ Ҝ к Ҟ ҟ
04A0 Ҡ ҡ Ң ң Ҥ ҥ Ҧ ҧ Ҩ ҩ Ҫ ҫ Ҭ ҭ Ү ү
Кирилли
Открыть на от
Диапазон: 040
Количество си
Тип: алфавит
Языки: русски
болгарский
perlunicode
5 / 73

WHITE SMILING FACE

U+263A

"\x{263A}"

"\xE2\x98\xBA"

"\342\230\272"

SMILING FACE WITH HORNS

U+1F608

"\x{1F608}"

"\xF0\x9F\x98\x88"

"\360\237\230\210"

UTF

Unicode Transformation Format

Формат преобразования юникода

Способ представления символов Unicode в виде последовательности целых положительных чисел

- **UTF-8 (8-битный) endianness safe**
- **UTF-16 (16-битный) LE | BE**
- **UTF-32 (32-битный) LE | BE**

```
 Code Points   Bytes: 1st    2nd    3rd    4th

  U+0000..U+007F     00..7F
  U+0080..U+07FF     C2..DF 80..BF
  U+0800..U+0FFF     E0     A0..BF 80..BF
  U+1000..U+CFFF     E1..EC 80..BF 80..BF
  U+D000..U+D7FF     ED     80..9F 80..BF
  U+D800..U+DFFF     utf16 surrogates, not utf8
  U+E000..U+FFFF     EE..EF 80..BF 80..BF
 U+10000..U+3FFFF    F0     90..BF 80..BF 80..BF
 U+40000..U+FFFFF    F1..F3 80..BF 80..BF 80..BF
U+100000..U+10FFFF   F4     80..8F 80..BF 80..BF
```

Строки и байты

Символ (character)

```
"\x{1}" .. "\x{10FFFF}"
chr(1)  .. chr(0x10FFFF)
```

Байт (символы 0..255)

```
"\x00" .. "\xff"
"\000" .. "\377"
chr(0) .. chr(255)
```

Октет - 8 бит данных

Строки и байты

Бинарные данные - строка из байт

```
my $bytes = "123";
my $bytes = "\001\002\377";
my $bytes = "\xfe\xff";
```

Строка - строка из символов (codepoints)

```
use utf8;
my $string = "Ёлка";#\x{401}\x{43b}\x{43a}\x{430}
my $string = "\x{263A}";
```

Преобразование

Энкодинг (*encode*)

преобразование текста (строк, символов) в данные (байты, октеты)

Декодинг (*decode*)

преобразование данных (байт, октетов) в текст (строки символов)

Преобразование

```
use Encode qw(encode decode);

my $string = decode("utf-8", "\342\230\272");
 # "\x{263a}"

my $octets = encode("utf-8", "\x{263a}");
 # "\342\230\272"

my $cp866_octets =
    encode(
        "cp866",
        decode("cp1251",$win_octets)
    );
```

UTF8_FLAG

```
say utf8::is_utf8("\342\230\272"); # ''
my $string = decode("utf-8", "\342\230\272");
say utf8::is_utf8($string); # 1

say utf8::is_utf8("\x{263a}"); # 1
my $octets = encode("utf-8", "\x{263a}");
say utf8::is_utf8($octets); # ''

printf "U+%v04X\n", decode('utf8',"тест");
# U+0442.0435.0441.0442

say utf8::is_utf8("☺"); # ''

printf "U+%v04X\n", "☺";
# U+00E2.0098.00BA
```

use utf8;

директива `use utf8` **"выполняет"**
`decode('utf8',<исходник>)`

```
use utf8;

say utf8::is_utf8("\342\230\272"); # ''

say utf8::is_utf8("\x{263a}"); # 1

say utf8::is_utf8("☺"); # 1
```

С флагом и без флага

```
$ perl -MDevel::Peek -E 'Dump "☺"'
SV = PV(0x7f8041804ae8) at 0x7f804182d658
  REFCNT = 1
  FLAGS = (PADTMP,POK,READONLY,pPOK)
  PV = 0x7f804140cf20 "\342\230\272"\0
  CUR = 3
  LEN = 16

$ perl -Mutf8 -MDevel::Peek -E 'Dump "☺"'
SV = PV(0x7fbf7a804b48) at 0x7fbf7b801f00
  REFCNT = 1
  FLAGS = (PADTMP,POK,READONLY,pPOK,UTF8)
  PV = 0x7fbf7a613920 "\342\230\272"\0 [UTF8 "\x{263a}"]
  CUR = 3
  LEN = 16
```

С флагом и без флага

```
$ perl -Mutf8 -MDevel::Peek -E 'Dump "\x{ff}"'
SV = PV(0x7fa153802948) at 0x7fa153005b00
  REFCNT = 1
  FLAGS = (PADTMP,POK,READONLY,pPOK)
  PV = 0x7fa152d06a10 "\377"\0
  CUR = 1
  LEN = 16

$ perl -Mutf8 -MDevel::Peek -E 'Dump "\x{100}"'
SV = PV(0x7fcdbc003548) at 0x7fcdbc02c100
  REFCNT = 1
  FLAGS = (PADTMP,POK,READONLY,pPOK,UTF8)
  PV = 0x7fcdbb707110 "\304\200"\0 [UTF8 "\x{100}"]
  CUR = 2
  LEN = 16
```

Поведение функций

```
my $test = "тест";
say length $test;
say uc $test;
say utf8::is_utf8 $test;
say ord(substr($test,0,1));
```

```
#
8
тест
''
209
```

```
use utf8;
my $test = "тест";
say length $test;
say uc $test;
say utf8::is_utf8 $test;
say ord(substr($test,0,1));
```

```
#
#
4
ТЕСТ
1
1090 (0x442)
```

perlunicode

@ARGV в UTF-8

```
$ perl -CA ...
```

или

```
$ export PERL_UNICODE=A
```

или

```
use Encode qw(decode_utf8);
BEGIN {
    @ARGV = map { decode_utf8($_, 1) } @ARGV;
}
```

perlunicode

STDIN, STDOUT, STDERR в UTF-8

Wide character in print at...

IO Layer `:utf8`

```
$ perl -CS ...
$ export PERL_UNICODE=S
```

```
binmode(STDOUT,':utf8');
open my $f, '<:utf8', 'file.txt';
use open qw(:std); # auto
```

perlunicode

Весь ввод/вывод в UTF-8

```
$ perl -CASD ... | perl -CS -CA -CD ...
```

```
$ export PERL_UNICODE=ASD
```

```
use open qw(:std :utf8);
use Encode qw(decode_utf8);
BEGIN{ @ARGV = map decode_utf8($_, 1),@ARGV; }
```

Ввод/вывод в октетах при UTF-8

IO Layer `:raw`

```
binmode($fh,':raw');

binmode(STDOUT,':raw');

open my $f, '<:raw', 'file.bin';

my $socket = accept(...);
binmode($socket,':raw');
```

Полезности

```
use utf8;
use Text::Unidecode;

say unidecode "\x{5317}\x{4EB0}"; # 北京
# That prints: Bei Jing

say unidecode "Это тест";
# That prints: Eto tiest
```

use charnames

```
use charnames qw(:full :short latin greek);
say "\N{MATHEMATICAL ITALIC SMALL N}"; # n
say "\N{GREEK CAPITAL LETTER SIGMA}"; # Σ
say "\N{Greek:Sigma}"; # Σ
say "\N{ae}"; # æ
say "\N{epsilon}"; # ε
say "\N{LATIN CAPITAL LETTER A WITH MACRON AND GRAV
$s = "\N{Latin:A WITH MACRON AND GRAVE}";
say $s;  # Ā`
printf "U+%v04X\n", $s; # U+0100.0300

use charnames ":alias" => {
    "APPLE LOGO" => 0xF8FF,
};
say "\N{APPLE LOGO}"; #
```

Casefolding

```
use feature "fc"; # perl v5.16+

# sort case-insensitively
my @sorted = sort {
    fc($a) cmp fc($b)
} @list;

# both are true:
fc("tschüß") eq fc("TSCHÜSS")
fc("Σίσυφος") eq fc("ΣΊΣΥΦΟΣ")
```

[Case Charts]

[perlunicode]

Documentation

perldoc

- perluniintro, perlunitut, perlunicook, perlunifaq, perlunicode, perluniprops

Modules

- Encode, Encode::Locale
- Unicode::UCD
- Unicode::Normalize, Unicode::CaseFold
- Unicode::GCString
- Unicode::LineBreak
- Unicode::Collate, Unicode::Collate::Locale

Регулярные выражения

Регулярные выражения

(*regular expressions*)

формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

[xkcd #208]

Credit card numbers

```
^(?:
    4[0-9]{12}(?:[0-9]{3})?           # Visa
|
    5[1-5][0-9]{14}                   # MC
|
    3[47][0-9]{13}                    # AmEx
|
    3(?:0[0-5]|[68][0-9])[0-9]{11}    # Diners
|
    6(?:011|5[0-9]{2})[0-9]{12}       # Discover
|
    (?:2131|1800|35\d{3})\d{11}       # JCB
)$
```

Сопоставление (m//)

```
my $string = "sample string";

$string =~  /sample/;
$string =~ m/sample/;
$string =~ m(sample);

my @a = $string =~ /sample/;
my $a = $string =~ /sample/;
if ($string =~ /sample/) { ... }

for (@samples) {
    /sample/;
    # $_ =~ /sample/;
    if (m/sample/) { ... }
    # if ($_ =~ /sample/) { ... }
}
```

Поиск и замена (`s///`)

```
my $string = "sample string";

$string =~ s/sample/item/;
$string =~ s{sample}{item};
$string =~ s{sample}
            (item);

my $count_of_replace =
    $string =~ s{sample}{item}g;

for (@samples) {
    s/sample/item/;
    # $_ =~ /sample/item/;
}
```

Транслитерация (`y///`, `tr///`)

```
my $str = "MiXeD CaSe StRiNg";

# ASCII lowercase;
$str =~ tr/A-Z/a-z/;
# mixed case string

# Change case
my $str = "MiXeD CaSe StRiNg";
$str =~ tr/A-Za-z/a-zA-Z/;
# mIxEd cAsE sTrInG

# ROT-13
$str =~ tr/A-Za-z/N-ZA-Mn-za-m/;
# zVkRq pNfR fGeVaT
```

Метасимволы

Символы, которые необходимо экранировать

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

Остальное можно использовать как есть

Классы символов (character classes)

```
[...]       # перечисление
/[abc]/       # "a" или "b" или "c"
/[a-c]/       # то-же самое
/[a-zA-Z]/    # ASCII алфавит

/[bcr]at/     # "bat" или "cat" или "rat"

[^...]      # отрицательное перечисление
/[^abc]/      # что угодно, кроме "a", "b", "c"
/[^a-zA-Z]/   # что угодно, кроме букв
```

Классы символов

```
`\d` - цифры. не только `[0-9]`
`\s` - пробельные символы `[\ \t\r\n\f]` и др.
`\w` - "буква". `[0-9a-zA-Z_]` и юникод
`\D` - не цифра. `[^\d]`
`\S` - не пробельный символ. `[^\s]`
`\W` - не "буква". `[^\w]`
`\N` - что угодно, кроме "\n"
`.`  - что угодно, кроме "\n" *
`^`  - начало строки **
`$`  - конец строки **
```

*** поведение меняется в зависимости от модификатора /s**

**** поведение меняется в зависимости от модификатора /m**

Квантификаторы

- `?` - 0 или 1 (`{0,1}`)
- `*` - 0 или более (`{0,}`)
- `+` - 1 или более (`{1,}`)
- `{x}` - ровно x
- `{x,y}` - от x до y включительно
- `{,y}` - от 0 до y включительно
- `{x,}` - от x до бесконечности

```
/^1?$/   # "" or "1"
/^a*$/   # "" or "a", "aa", "aaa", ...
/^\d*$/  # "" or "123", "11111111", ...
/^.+$/   # "1" or "abc", not ""

/^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}$/
    # "2015-10-14 19:35:01"
```

Захваты

`$1`, `$2`, `$3`, ...

```
$_ = "foo bar baz";

m/^(\w+)\s+(\w+)\s+(\w+)$/;
# $1 = 'foo';
# $2 = 'bar';
# $3 = 'baz';

m/^(\w(\w+))\s+((\w+))/;
#  1  2         34
# $1 = 'foo';
# $2 = 'oo';
# $3 = 'bar';
# $4 = 'bar';
```

Группы

`(...)` - **захватывающая группа**

`(?:...)` - **незахватывающая группа**

```
"a" =~ /^(?:a|b|cd)$/;    # match
"b" =~ /^(?:a|b|cd)$/;    # match
say $1; # undef
"ax" =~ /^(?:a|b|cd)$/;   # no match

"a" =~ /^(a|b|cd)$/;    # match
say $1; # a
"b" =~ /^(a|b|cd)$/;    # match
say $1; # b
"ax" =~ /^(a|b|cd)$/;   # no match
say $1; # undef
```

Группы

`(?<name>...)`
`(?'name'...)` **- захватывающая именованная группа**

```
"abc" =~ /^(.)(.)/;
say "first: $1; second: $2";
# first: a; second: b

"abc" =~ /^(?<first>.)(?<second>.)/;
say "first: $+{first}; second: $+{second}";
# first: a; second: b
```

Оглядывания

`(?=...)` - 0W+ вперёд

`(?!...)` - 0W- вперёд

`(?<=...)` - 0W+ назад

`(?<!...)` - 0W- назад

```
$_ = "foo bar baz";

say s{(\w+)(?=\s+)}{$1,}r; # foo, bar, baz
say s{(\s+)(?!bar)}{_}r; # foo bar_baz

say s{(?<= )(\w+)}{:$1}rg; # foo :bar :baz
say s{(?<! )(\w\w\w)}{[$1]}rg; # [foo] bar baz
```

Модификаторы

`/s` **(single line) - включает в** `.` **всё**

```
"\n" =~ /^.$/; # no match
"\n" =~ /^.$/s; # match

my $s = "line1\nline2\n";

$s =~ /line1.line2/; # no match
$s =~ /line1.line2/s; # match
```

Модификаторы

`/m` **(multiline)**

- `^` **начало каждой строки**
- `$` **конец каждой строки (до \n)**

```
my $s = "sample\nstring";

$s =~ /^(.+)$/;     # no match
$s =~ /^(.+)$/m;    # "sample"
$s =~ /^(.+)$/ms;   # "sample\nstring"

$s =~ /^string/;    # no match
$s =~ /^string/m;   # matches "string"
```

Модификаторы

`/i` **(case insensitive)**

```
my $s = "sample\nstring";

$s =~ /SAMPLE/;     # no match
$s =~ /SAMPLE/i;    # "sample"

# Unicode!

"tschüß"   =~ /TSCHÜSS/i    # match. ß ↔ SS
"Σίσυφος" =~ /ΣΊΣΥΦΟΣ/i    # match. Σ ↔ σ ↔ ς
```

Модификаторы

`/x` **(eXtended regexp)**

```
$hexdig =~ m{
    ^ # begin of string
    (?:
        [0-9] # it could be digit
        |     # or
        [a-f] # letters from a to f
    )+ # several times
    $ # end of string
}sx;
```

Модификаторы

`/g` **(global)**

```
my $s = "aaaa";
$s =~ s/a/b/;  # "baaa"
$s =~ s/a/b/g; # "bbbb"

@a = $a =~ /(.)/; # ('a')
@a = $a =~ /(.)/g; # ('a','a','a','a')

my $string = '~!@#$%^&*()';
$string =~ s{(.)}{\\$1}g;
# \~\!\@\#\$\%\^\&\*\(\)
```

Модификаторы

`/e`, `/ee` **(eval, double eval)**

```
my $string = '~!@#$%^&*()';

$string =~ s{(.)}{
    sprintf("U+%v04x;",$1)
}ge;
#U+007e;U+0021;U+0040;U+0023;U+0024;U+0025;
#U+005e;U+0026;U+002a;U+0028;U+0029;

my $nums = "0x123 123 0xff";
$nums =~ s{0x([\da-f]+)}{ hex($1) }ge;
say $nums; # 291 123 255
```

Модификаторы

`/e`, `/ee` **(eval, double eval)**

не печатает

```
cat "test... test... test..." | perl -e '$??
s:;s:s;;$?::s;;=]=>%-{%#(/|}<&|`{;;y; -/:-@[-`{-
};`-{/" -;;s;;$_;see;'
```

Модификаторы

`/e`, `/ee` **(eval, double eval)**

```
$?
    ?
        s:;s:s;;$?:
    :
        s;;=]=>%-{%#(/|}<&|`{;
    ;
y; -/:-@[-`{-};`-{/" -;;
s;;$_;see;
```

Модификаторы

`/e`, `/ee` **(eval, double eval)**

```
$?
    ?
        s/;s/s;;$?/
    :
        $_ = '=]=>%-{%#(/|}<&|`{'
    ;
tr( -/:-@[-`{-})
  (`-{/" -);

s//$_/see;
```

Модификаторы

`/e`, `/ee` **(eval, double eval)**

```
$?
    ?
        s/;s/s;;$?/
    :
        $_ = '=]=>%-{%#(/|}<&|`{'
    ;
tr( -/:-@[-`{-})
  (`-{/" -);
say $_; # system"echo -rf /"
s//$_/see;
# match empty string in $_
# replace it with eval(eval( '$_' ))
# eval '$_' gives 'system"echo -rf /"'
# eval 'system"echo -rf /"' gives ...
```

Модификаторы

`/a`, `/aa` **(ASCII-safe) (**`\d`, `\s`, `\w`**)**

```
use utf8;
use charnames ':full';
my $nums = "০১২৩";
$nums =~ /\d/; # match
$nums =~ /\d/a; # no match

my $str = "\N{KELVIN SIGN}";
say $str =~ /K/i; # match
say $str =~ /K/ai; # match
say $str =~ /K/aai; # no match
```

Квантификаторы и жадность

`?` - 0 или 1 (`{0,1}`)

`*` - 0 или более (`{0,}`)

`+` - 1 или более (`{1,}`)

`{x}` - ровно x

`{x,y}` - от x до y включительно

`{,y}` - от 0 до y включительно

`{x,}` - от x до бесконечности

quantifier `?` - минимально

quantifier `+` - без отката

Квантификаторы и жадность

```
say "bc"     =~ /^(a*)b/;   # match, ""
say "abc"    =~ /^(a*)b/;   # match, "a"
say "aabc"   =~ /^(a*)b/;   # match, "aa"
say "aaabc"  =~ /^(a*)b/;   # match, "aaa"

say "aaabc"  =~ /^(a*)/;    # match, "aaa"
say "aaabc"  =~ /^(a*?)/;   # match, ""
say "aaabc"  =~ /^(a*?)a/;  # match, ""
say "aaabc"  =~ /^(a*?)ab/; # match, "aa"

say "aaabc"  =~ /^(a*+)/;   # match, "aaa"
say "aaabc"  =~ /^(a*+)b/;  # match, "aaa"
say "aaabc"  =~ /^(a*+)ab/; # no match
```

Квантификаторы и жадность

```
say "bc"    =~ /^(a+)b/;   # no match
say "abc"   =~ /^(a+)b/;   # match, "a"
say "aabc"  =~ /^(a+)b/;   # match, "aa"
say "aaabc" =~ /^(a+)b/;   # match, "aaa"

say "aaabc" =~ /^(a+)/;    # match, "aaa"
say "aaabc" =~ /^(a+?)/;   # match, "a"
say "aaabc" =~ /^(a+?)a/;  # match, "a"
say "aaabc" =~ /^(a+?)ab/; # match, "aa"

say "aaabc" =~ /^(a++)/;   # match, "aaa"
say "aaabc" =~ /^(a++)b/;  # match, "aaa"
say "aaabc" =~ /^(a++)ab/; # no match
```

Квантификаторы и жадность

```
say "bc"    =~ /^(a{1,2})b/; # no match
say "abc"   =~ /^(a{1,2})b/; # match, "a"
say "aabc"  =~ /^(a{1,2})b/; # match, "aa"
say "aaabc" =~ /^(a{1,2})b/; # no match

say "aaabc"  =~ /^(a{1,2})a/;  # match "aa"
say "aaabc"  =~ /^(a{1,2}?)a/; # match "a"
say "aabc"   =~ /^(a{1,2}?)b/; # match "aa"
```

Backreferencing

`\gN` **или** `\N` **или** `\g{-N}`

```
for (
    q{some with "quoted value" string},
    q{some with 'quoted " value' string},
) {
    say $2 if m{(["'])([^\g1]*)\g1};
}
# quoted value
# quoted " value

for ('e66e', 'f99f', 'z87z' ) {
    say $1 if m{(([a-z])(\d)\g{-1}\g{-2})}x;
}
#e66e
#f99f
```

Global match

`/g`, `/c`, `\G` и `pos()`

```
$_ = "abcd";
while (/(.)/g) {
    say $1, " ", pos($_);
    # a 1
    # b 2
    # c 3
    # d 4
}
say $1, " ", pos($_);
# undef, undef
```

Global match

`/g`, `/c`, `\G` и `pos()`

```
$_ = "abcd";
while (/(.)/gc) {
    say $1, " ", pos($_);
    # a 1
    # b 2
    # c 3
    # d 4
}
say $1, " ", pos($_);
# undef, 4
```

Global match

`/g`, `/c`, `\G` и `pos()`

```
$_ = "abcdxcdcd";
while (/\G(.)/gc) {
    my $key = $1;
    my $pos = pos($_);
    if (/\Gcd/gc) {
        say "the key before cd is $key at $pos";
    } else {
        say "no cd next after $key";
    }
}
# no cd next after a
# the key before cd is b at 2
# the key before cd is x at 5
# no cd next after c
# no cd next after d
```

Классы символов Unicode

`\p{Category}` - совпадение с категорией
`\P{Category}` - исключение категории
`\N{SYMBOL NAME}` - точное имя (see charnames)

```
"UPPER" =~ /\p{IsUpper}/; # match
"UPPER" =~ /\P{IsUpper}/; # no match
"UPPER" =~ /\p{IsLower}/; # no match
"UPPER" =~ /\P{IsLower}/; # match

say q{«string"with'quotes»} =~
    s/\p{Quotation Mark}+/ /rg;
# ' string with quotes '
```

Отладка регулярных выражений

```
use re 'debug';
```

```
perl -Mre=debug -E '"aaabc"   =~ /^(a{1,2}?)ab/;'
```

```
Compiling REx "^(a{1,2}?)ab"
Final program:
   1: BOL (2)
   2: OPEN1 (4)
   4:   MINMOD (5)
   5:   CURLY {1,2} (9)
   7:     EXACT <a> (0)
   9: CLOSE1 (11)
  11: EXACT <ab> (13)
  13: END (0)
```

Отладка регулярных выражений

```
Guessed: match at offset 0
Matching REx "^(a{1,2}?)ab" against "aaabc"
   0 <> <aaabc>                |  1:BOL(2)
   0 <> <aaabc>                |  2:OPEN1(4)
   0 <> <aaabc>                |  4:MINMOD(5)
   0 <> <aaabc>                |  5:CURLY {1,2}(9)
                     EXACT <a> can match 1 times out of 1...
   1 <a> <aabc>                |  9:  CLOSE1(11)
   1 <a> <aabc>                | 11:  EXACT <ab>(13)
                       failed...
                     EXACT <a> can match 1 times out of 1...
   2 <aa> <abc>                |  9:  CLOSE1(11)
   2 <aa> <abc>                | 11:  EXACT <ab>(13)
   4 <aaab> <c>                | 13:  END(0)
Match successful!
```

Список документации

- perlre
- perlrequick
- perlretut
- perlrecharclass
- re
- pos

Домашнее задание

Реализовать с помощью регулярных выражений парсер (синтаксический анализатор) формата данных JSON (http://json.org). На выходе необходимо получить структуру, аналогичную возвращаемой модулем JSON::XS:

```
use DDP;
sub decode_json {
    my $data = shift;
      ... # To be done
    return $ref;
}
my $data = do { # чтение файла
    open my $f,'<:raw',$ARGV[0]
    or die "open `$ARGV[0]' failed: $!";
    local $/; <$f>
};

my $struct = decode_json($data);
p $struct;
```

Пример файла в виде JSON

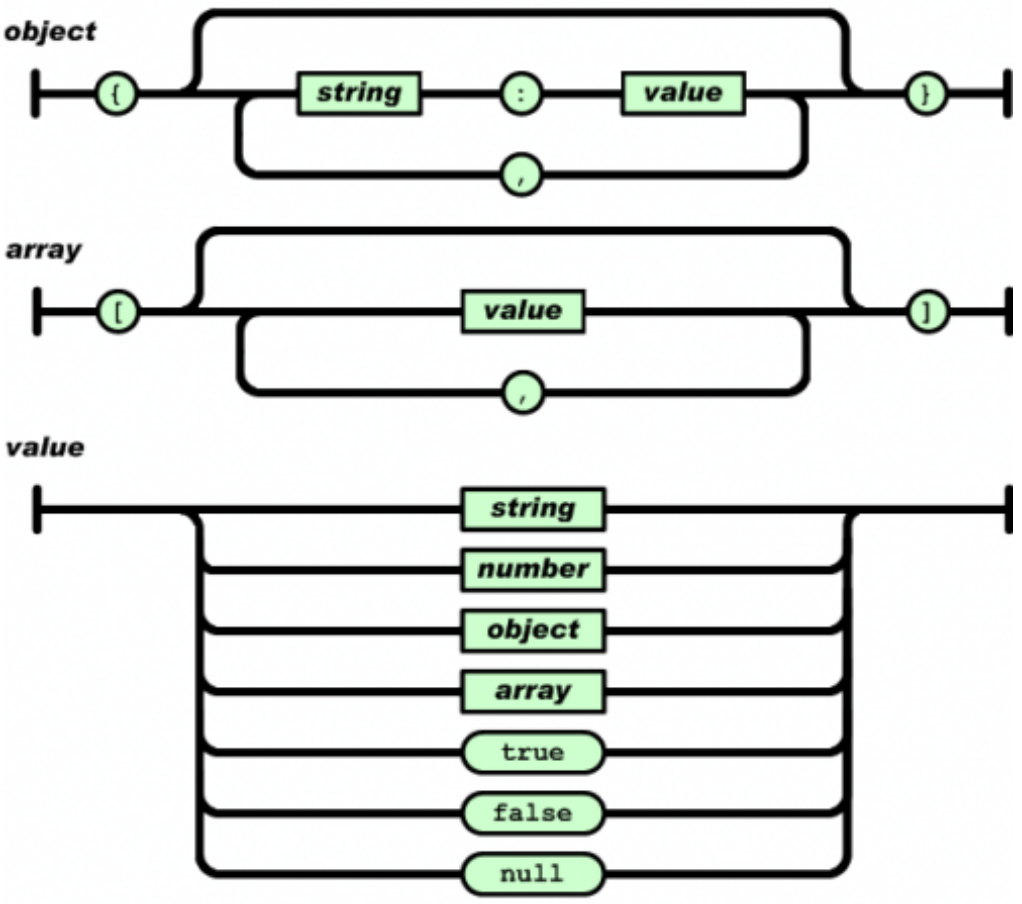
data.json:

```
{
  "key1": "string value",
  "key2": -3.1415,
  "key3": ["nested array"],
  "key4": { "nested": "object" },
}
```

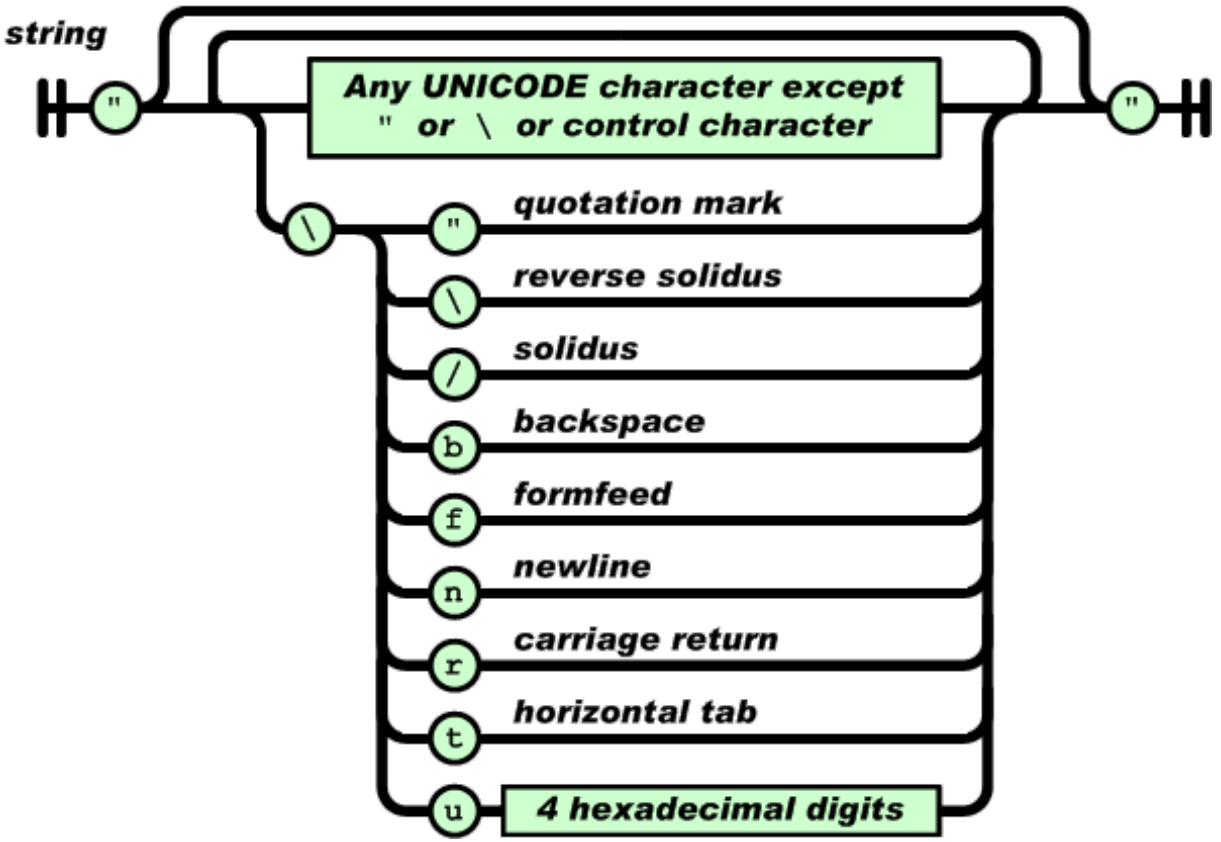
Тест с такими данными

```
> perl json.pl data.json

\ {
    key1   "string value",
    key2   -3.1415,
    key3   [
        [0] "nested array"
    ],
    key4   {
        nested   "object"
    }
}

>
```

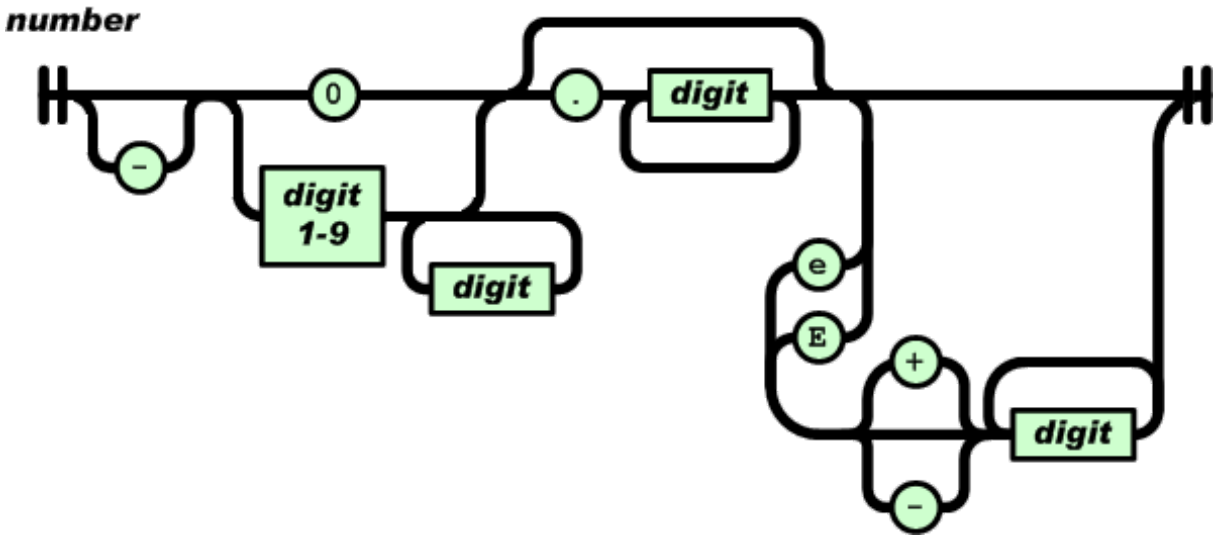
Диаграмма структуры JSON



JSON-строка


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

JSON-число



Домашнее задание: требования

Минимальный удовлетворительный набор:

1. **Поддержать объект, массив, строку, число**

```
{ "k": "v" }, [1,2,3], "string", 123
```

2. **В строке поддержать последовательности \", \n, \uXXXX**

```
"my string with \"\u410\n"
```

3. **В числе поддержать унарный минус и десятичную точку**

```
0, 0.1234, -17000
```

4. **Разрешается пропускать "висящую" запятую**

```
[ 1, 2, 3, ]
```

Не обязательно парсить всё одним регэкспом

__END__

Благодарю за внимание!

Mons Anderson <mons@cpan.org>