# PB173 Perl

## 01 Úvod

Roman Lacko <xlacko1@fi.muni.cz>

# Obsah

# Úvod

> Python is executable pseudocode. Perl is executable line noise.
>
> — Bruce Eckel

## Ciele predmetu

- Naučiť sa nový jazyk!
- ~~Potvrdiť~~ vyvrátiť zlú povesť Perlu

### Predpoklady

- IB111 Základy programování
- Znalosť iného skriptovacieho jazyka (Python, Ruby)
- Základy shellu
- Git

## Organizácia cvičení

- 3 cvičenia × 4 bloky
- Ukážky (`exNNt-*.pl`)
- Príklady (`exNNp-*.pl`)
- Domáce úlohy (`hw*.pl`)

### Študijné materiály

- Slidy v študijných materiáloch IS MU
- Zadania cvičení v GitLab FI

## Hodnotenie

Nutné splniť **všetky** nasledujúce podmienky:

- Vypracovať 8 ľubovoľných úloh z rôznych týždňov.
  - Termín odovzdania je týždeň po zadaní.
  - Na každú úlohu bude jedno opravné odovzdanie.
- Nanajvýš 2 neospravedlnené neúčasti.
  - Ďalšie dve absencie je možné nahradiť vypracovaním **všetkých** úloh v danom týždni.

# Prečo Perl?

- Prakticky zameraný jazyk
- Rôzne úrovne abstrakcie a štýlov
- Veľké množstvo rozširujúcich balíkov (CPAN)

## Použitie

- Rýchle skriptovanie, prototypovanie
- Webové aplikácie (CGI skripty)
- Práca s DB (Perl DBI)
- „Lepidlový“ jazyk
- Spracovanie textov

IMDb, DuckDuckGo, Bugzilla, AoC, IS MU

## Write-only paradigm?

Čo robí nasledujúci program?

```c
#include <stdio.h>
#include <stdlib.h>
#define B 69455037737123477 54LL
#define I 585983823119196245 9LL
int main(int b,char**i){
    long long n=B,a=I^n,r=(a/b&a)>>4,y=atoi(*++i),
        _=(((a^n/b)*(y>>0)|y>>7)&r)|(a^r);
    printf("%.8s\n",(char*)&_);
}
```

## Write-only paradigm?

Čo robí nasledujúca funkcia?

```
from functools import reduce

def a(n):
    return reduce((lambda r,x: r-set(range(x**2,n,x)) if (x in r) \
        else r), range(2,int(n**0.5)), set(range(2,n)))
```

## Write-only paradigm?

Napísať nepochopiteľný kód je možné v **každom** jazyku.
V Perli je to len trochu jednoduchšie.

- Vhodné či nevhodné prístupy v ukážkach kódov.
- Dôraz na čistotu kódu pri hodnotení úloh.

Píšte kód konzistentne **konzistentne** a snažte sa dodržiavať štýl v ukážkach.

# História a budúcnosť jazyka

## Larry Wall

- *1954, Los Angeles
- `patch(1)`, Perl
- Dvojnásobný víťaz *IOCCC*.
- Lingvista
- Silný veriaci (`bless`, *apokalypsy*)

## Pe(a)rl 1 (1987)

- C (syntax výrazov, konštrukcie)
- Shell (`$var`, `-f`, `-t`, *heredocs*)
- Sed (`m//`, `s///`, `qr//`)
- … a ďalšie

> Perl officially stands for *Practical Extraction and Report Language*, except when it doesn't.
>
> …
>
> Perl actually stands for *Pathologically Eclectic Rubbish Lister*, but don't tell anyone I said that.
>
> — man perl

## Perl 5 (1994)

- Objekty
- Moduly

`v5.34` — Aktuálna verzia
`v5.32` — Verzia na Aise

## Perl 6 (2015), Raku (2019)

- Úplne nový a samostatný jazyk

## Perl 7 (?)

- Plánovaný nástupca Perl 5
- Zahodenie veľmi starých nepotrebných častí
- Stabilizácia nových vlastností (napr. signatúry)

# Hello, world!

Stiahnite si materiály podľa návodu:
https://gitlab.fi.muni.cz/xlacko1/pb173-perl

ex01t-hello.pl

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

use v5.32;
use warnings;

say "Hello, World!";
```

```
$ perl -Mv5.32 -e 'say "Hello, World!"'
```

## Základy

```
say VALUE, ...;
```

- Vypíše hodnoty v parametroch a ukončí riadok.

```
print VALUE, ...;
```

- Vypíše hodnoty v parametroch bez ukončenia riadka.
- `say A, B, C` ≈ `print A, B, C, "\n"`

```
printf FORMAT, ...;
```

- Vypíše formátovaný reťazec.
- Formátovacie značky ako `printf(3)`, a navyše nejaké ďalšie.

## Základy

Pripomeňte si funkciu `printf` z C.

Vyskúšajte si: `ex02p-name.pl`

Pohodlnejšiu *interpoláciu reťazcov* si ukážeme neskôr.

## Základy

Číselné konštanty a operátory fungujú **skoro** ako v C:

ex03t-numbers.pl

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

use v5.32;
use warnings;

say 42;
say 1 + 2;
say 1 + 2 * 3;
say 5 ** 2;
say 7 % 3;
say 1 << 7;

say 3.1415;
say 1 / 2;
```

## Základy

Reťazce sa spájajú operátorom `.`:

ex04t-strings.pl

```
say "Hello" . ", World" . "!";
```

## Základy

- `+`, `-` atď. sú *aritmetické* operátory
- `.` je *reťazcový* operátor

Čo sa stane, ak zmiešame typy operandov?

`ex05t-mixing.pl`

```
say "1" +  2;
say "1" + "2";
say  3  .  4;
say  3  . "4";
```

```
say 3 + "";
say 7 - "3 days";
say 2 * "Rust";
say 4 / "OMG Perl";
```

# Základy

## Pozorovanie

- Niektoré operátory vynútia koerciu svojich argumentov.
- Reťazce sa prevádzajú na čísla mechanizmom podobným `atoi(3)`.
- `use warnings` varuje pred prevodom reťazca,
  ktorý nereprezentuje číslo.

# Základy

## Zátvorky?

Zátvorky vo výrazoch vynucujú prioritu vyhodnotenia:

```
1 + 2 * 3       # → 7
(1 + 2) * 3     # → 9
```

Niektoré jazyky ich tiež *vyžadujú* pri volaní funkcií:

memgrind.c

```
fprintf(stderr, "memgrind: Invalid pointer\n");
```

Vyskúšajte si, ako sa chová Perl:

ex06p-parenthesis.pl

# Základy

## Zátvorky

Je chovanie príkladu `ex06p-parenthesis.pl` divné?

`main.c`

```
int pow(int n);
// ...
pow (1 + 2) * 3;
```

## Konvencia

- *Zabudované* funkcie by sa mali volať bez zátvoriek.
- Pri jednoduchých výrazoch to zlepšuje prehľadnosť.

Zátvorky však píšte, ak je výraz v argumente zložitejší a nedá sa napísať inak, alebo ak si nie ste istí poradím vyhodnocovania.

# Skaláry

Perl má tri základné „typy“ premenných:

**`$NAME`**
: Skaláry (čísla, reťazce, *referencie*, ...)

**`@NAME`**
: Polia

**`%NAME`**
: Asociatívne polia („hashe“)

`$`, `@` a `%` a volajú *sigils*.

## Skalárne premenné

Premenné deklarujeme pomocou `my`.
Pred názvom *skalárnej* premennej uvádzame `$`.

ex09t-scalars.pl

```
my $name = "Perl";
my $version = 5;

say $name, ": ", $version;
```

`my` zavádza *lexikálnu* premennú, ktorá žije od momentu deklarácie po koniec rozsahu.

## Skalárne premenné

Môžeme deklarovať aj viac premenných naraz, sú však nutné zátvorky.

`ex07t-scalars.pl`

```
my ($a, $b) = (0, 1);
```

Čo sa stane, ak zátvorky vynecháte?

### Neinicializované premenné

```
my $result;
```

- Špeciálna hodnota `undef` (≈ `None` v Pythone).
- Predikátový operátor `defined $variable`

## Interpolácia reťazcov

Namiesto

```
my $message = "Hello, " . $name . "!";
```

môžeme písať

```
my $message = "Hello, $name!";
my $message = "Hello, ${name}!";
```

Čo vypíše nasledujúci príklad?

```
my $base = "source";
say "$base_name.txt";
```

# Výrazy

## Aritmetické operátory

```
+A
A + B
A * B
A % B
A++
A--
```

```
-A
A - B
A / B
A ** B
++A
--A
```

```
X = X ∘ Y → X ∘= Y
```

## Bitové operátory

```
~A          A & B       A | B
A ^ B       A << B      A >> B
```

## Reťazcové operátory

```
S . T
```

## Ternárny operátor

```
CONDITION ? IF_TRUE : IF_FALSE;
```

## Logické hodnoty

- Perl nemá konštanty `true` ani `false`
- Nepravdivé hodnoty: `undef`, `""`, `0`, `"0"`
- Všetky ostatné hodnoty sú pravdivé

Aká je logická hodnota `0.0`, `0E0` alebo `"0 but true"`?

## Relačné operátory

| | | | | | | |
|---|---|---|---|---|---|---|
| Čísla | `<` | `>` | `<=` | `>=` | `==` | `!=` |
| Reťazce | `lt` | `gt` | `le` | `ge` | `eq` | `ne` |

Porovnania je možné reťaziť, napr. `0 < $var < 10`

## Logické operátory

| | | |
|---|---|---|
| `!` | `&&` | `\|\|` |
| `not` | `and` | `or` |

Preferujte operátory v prvom riadku, ostatné majú *nižšiu* prioritu:

```
say 1 &&  2;    # → say(1 && 2)
say 1 and 2;    # → say(1) and 2
```

## Komparátory

| | |
|---|---|
| `<=>` | `cmp` |

# Príkazy

## Priradenie

ex10t-statements.pl

```
my $x;
# ...
$x = 2 * $y;
$x = $y = $z;
```

Priradenie funguje aj na *zoznamoch* premenných:

```
($x, $y) = ($y, $x);
```

Zoznamy a polia budú vysvetlené neskôr.

## Podmienky

ex08t-statements.pl

```
if (CONDITION) {
    STATEMENTS...;
} elsif (CONDITION) {
    STATEMENTS...;
} else {
    STATEMENTS...;
}
```

Zátvorky `( )` aj `{}` sú povinné.

Časti `elsif` aj `else` sú nepovinné.

## Podmienky

ex08t-statements.pl

```
unless (CONDITION) {
    STATEMENTS;
}
```

Ekvivalent `if` s negáciou podmienky.
*Radšej nepoužívajte.* A už vôbec nie s `else`.

```
unless ($a && ($b || !$c)) {
    STATEMENTS;
} else {
    # ... /o\
}
```

# Postfixové podmienky

Špecialita Perlu

`ex10t-control.pl`

```
STATEMENT if CONDITION;
STATEMENT unless CONDITION;
```

- Používajte **veľmi** opatrne.
- `STATEMENT` aj `CONDITION` by mali byť veľmi jednoduché, dlhé podmienky napíšte civilizovane.

Vhodné na ošetrenie vstupných podmienok (*Early Return*).

## Switch

```
use experimental 'switch';

given (EXPRESSION) {
    when (MATCH) {
        STATEMENT;
    }

    STATEMENT
        when MATCH;

    default {
        STATEMENT;
    }
}
```

## Switch: Výraz vo `when`

Perl sa snaží „uhádnuť“, čo chceme testovať. Pravidlá sú trochu zložité, preto je táto konštrukcia stále experimentálna.

Test môže byť prakticky čokoľvek, napríklad

- `SCALAR`: priamy test so skalárnou hodnotou
- `[A..B]`: test rozsahu
- `A < $_`: explicitný test porovnania
- regulárny výraz
- referencia na funkciu
- výrazy spojené operátorom `&&` alebo `and` (radšej nepoužívajte)

Premennú `$_` Perl definuje v danom rozsahu sám.
Viac o tejto premennej neskôr.

## Switch: `break`?

V blokoch pre `when` sa môžu nachádzať kľúčové slová:

- `break` ukončí `given`
  (implicitne na konci každého `when` bloku)
- `continue` implementuje *fall through*

```
given ($value) {
    when ([0..5]) {
        break if $_ == 3;
        say "When block";
        continue; # Explicit fall-through
    }
    default {
        say "Default block";
    }
}
```

## Cykly

ex08t-statements.pl

```
while (CONDITION) {
    STATEMENTS;
}
```

```
until (CONDITION) {
    STATEMENTS;
}
```

Riadenie cyklu:

- `last` — ukončí cyklus (≈ `break`)
- `next` — ďalšia iterácia (≈ `continue`)

Za cyklom môže byť `continue BLOCK`, ktorý sa vykoná pred každou iteráciou, nepoužíva sa často.

## Pomenované cykly

```
OUTER:
while (CONDITION) {
    while (OTHER_CONDITION) {
        last OUTER if STOP_CONDITION;
        ...
    }
}
```

## Cykly

```
for (EXPRESSION; CONDITION; INCREMENT) {
    STATEMENTS;
}
```

Namiesto `for` je možné povedať aj `foreach`

### Cyklus cez rozsah hodnôt (inkluzívne)

```
foreach my $v (A..B) {
    STATEMENTS;
}
```

## `do` blok

```
do {
    STATEMENTS;
};
```

- Vráti hodnotu posledného príkazu v bloku.
- Možné spojiť s postfixovým `while` a `until`:

```
do {
    STATEMENTS;
} while CONDITION;
```

V takomto cykle nie je možné použiť `last`, `next` ani `redo`!

Ďalšie cykly nabudúce.

# Funkcie

Perl volá funkcie „subrutiny“, preto `sub`:

```
use experimental 'signatures';

sub add($a, $b) {
    return $a + $b;
}
```

Pragma `signatures` umožní predávať argumenty v pomenovaných premenných ako u iných príčetných jazykov. Ako sa predávajú parametre bez nej uvidíte neskôr.

## Východzie hodnoty

```
sub add($a, $b = 0) {
    return $a + $b;
}

add(2)   # → 2
```

## Nevyužité argumenty

```
sub dummy($a, $, $=) {
    ...;
}
```

V iných jazykoch sa často používa _.
Prečo nemôže mať Perl tiež _ alebo $_?