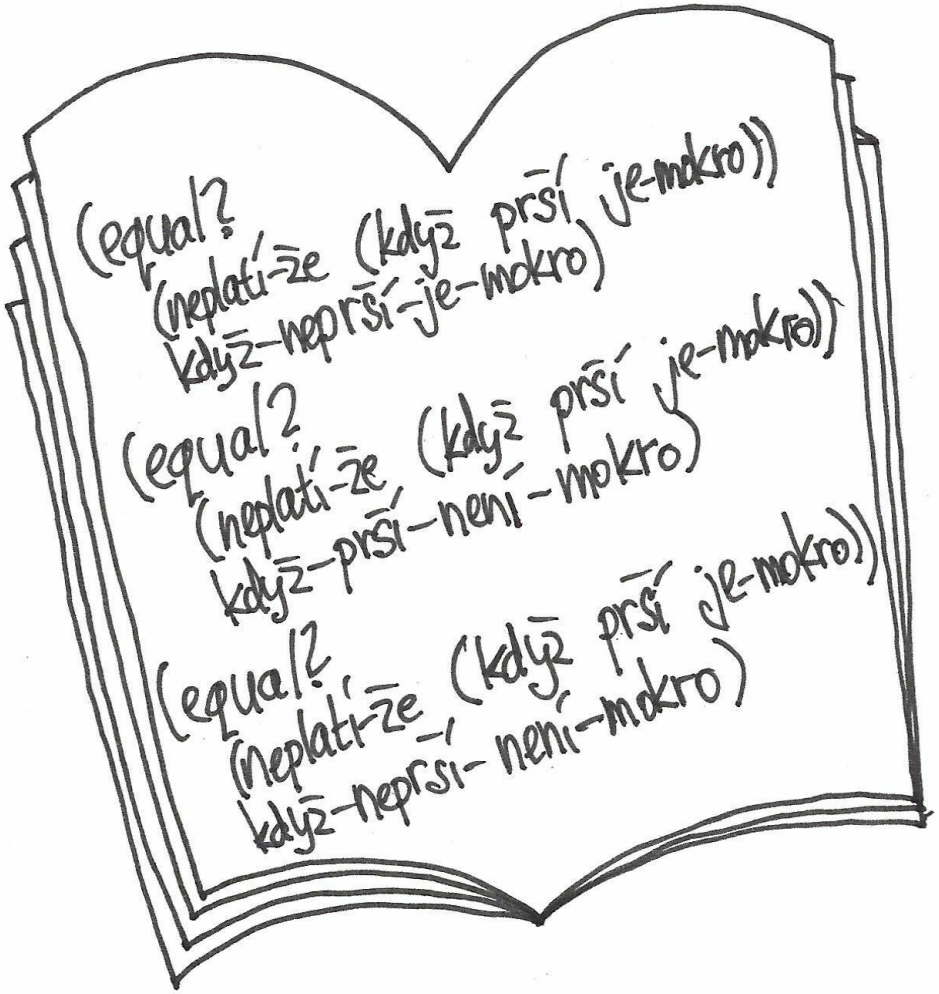
(equal?
(neplatí-že (když prší je-mokro))
když-neprší-je-mokro)
(equal?
(neplatí-že (když prší je-mokro))
když-prší-není-mokro)
(equal?
(neplatí-že (když prší je-mokro))
když-neprší-není-mokro)

Kouzelné čtení

Jazyk je jeden ze způsobů, jak se dají sdílet myšlenky. Použije se při mluvení a poslouchání a psaní a čtení. To, že kouzelná hůlka dokáže vykonat věty zapsané v některém z jazyků, neovlivňuje možnost sdílet v jazyce myšlenky. Vykonávání kouzelných vět kouzelnou hůlkou ale souvisí s tím, jak se dá s myšlenkami vyjádřenými v kouzelném jazyce pracovat, protože dost práce udělá právě kouzelná hůlka.

Jen pro připomenutí - kouzelný jazyk *Racket* má dva druhy slov: podstatná jména a slovesa. Podstatná jména ukrývají nějakou informaci. Třeba "kniha", "kouzelná", 0 a 3.14 jsou podstatná jména kouzelného jazyka. Slovesa říkají, co se má s podstatnými jmény stát. Třeba + a **string-append** jsou slovesa kouzelného jazyka. Každá kouzelná věta začíná závorkou (a hned po ní následuje sloveso. Potom jsou podstatná jména oddělená mezerami a úplně nakonec zase závorka).

Několik myšlenek o nekouzelném světě

A teď už několik myšlenek o nekouzelném světě zapsaných ve větách kouzelného jazyka.

Rekurze, iterace, nebo cyklus?

Rekurze, iterace i cyklus souvisí s opakováním.

Rekurze je, když kouzelná hůlka vykonává kouzelnou větu, ve které potřebuje vykonat jinou kouzelnou větu začínající stejným kouzelným slovesem.

Iterace je, když kouzelná hůlka opakuje kouzelné věty, ve kterých se mění informace ukryté v některých kouzelných podstatných jménech.

Cyklus je, když kouzelná hůlka opakuje kouzelné věty, dokud jedna z těch kouzelných vět platí.

Funkce vyššího řádu

Funkce vyššího řádu (higher order function) jsou funkce, které přijímají funkce jako argumenty a/nebo je funkce jejich návratová hodnota. Jinými slovy - kouzelná věta je vyššího řádu, když kromě prvního kouzelného slovesa je nějaké další slovo kouzelné věty také kouzelné sloveso a/nebo je kouzelné sloveso výsledek vykonání kouzelné věty.

Uzávěry a anonymní funkce

Uzávěr (closure) je, když si funkce pamatuje a použije informaci dostupnou při definici funkce, která není argument funkce.

Anonymní (lambda) funkce je funkce, která nemá jméno.

Abstrakce

Abstrakce je, když se více problémů dá vyřešit jedním způsobem. To znamená, že ty problémy mají něco společného. Liší se pak v konkrétních detailech.

Co třeba účastnící zájezdu? Který je nejmladší? A který bude první v abecedním seznamu? To společné je, že je potřeba účastníky seřadit. To konkrétní je podle čeho - datum narození nebo jméno.

CC BY-SA 4.0 neprogramator.cz

Čarodějnický žargon

Občas může neprogramátor zaslechnout čaroděje říkat slova, která sice nejsou kouzelná, ale stejně jim nerozumí. Občas jsou podobná slovům, které zná, a občas samotná slova dávají smysl, ale sousloví už ne.

Imperativní programování

Imperativní programování je založené na příkazech. Programátor říká počítači, co přesně má počítač vykonat. Příkazy mohou změnit stav vykonávaného programu (a světa obecně). Tomu, když se to stane, se říká *vedlejší jev* (side effect).

Procedurální programování je imperativní; vylepšuje strukturu zdrojového kódu tím, že příkazy rozděluje na menší části, které pojmenuje. Pojmenované části mohou být součástí jiných pojmenovaných částí a je možné je vykonávat znovu a znovu.

Objektově orientované programování (OOP) je impeprativní; vylepšuje strukturu zdrojového kódu tím, že schovává příkazy a části stavu programu do *objektů*. Tím vytváří hierarchii. Objekty mezi sebou komunikují přes *rozhraní* (interface, API), které sestává z dvou částí: (1) *metody*, které se na objekt *volají* a (2) *parametry* (argumenty), které se metodám *předávají*. Objekty v programu často představují zjednodušenou verzi objektů reálného světa a mohou obsahovat další objekty.

Deklarativní programování

Deklarativní programování je založené na vztazích. Programátor říká počítači co s čím souvisí a jak.

Funkcionální programování je deklarativní; konkretizuje jej na matematické funkce. Matematická funkce nemůže nic změnit, protože jen popisuje vztah. Říkáme, že je to *čistá funkce* (pure function).

Logické programování je deklarativní; programátor popíše logické vztahy a nechá počítač, aby odpověděl na jeho otázku.

Krátká matematická myšlenka

```
(define (Pythagorova-věta a b c) ; a, b jsou odvěsny, c přepona
  (= (* c c) (+ (* a a) (* b b))))

(define (vypočítej-přeponu-pro-odvěsny a b)
  (sqrt (+ (* a a) (* b b))))

(define (vypočítej-odvěsnu-pro-přeponu-a-odvěsnu c a)
  (sqrt (- (* c c) (* a a))))
```

Krátká chemická myšlenka

```
(define (oxidační-číslo->koncovka oxidační-číslo)
  (cond ((= 1 oxidační-číslo) "-ný")
        ((= 2 oxidační-číslo) "-natý")
        ((= 3 oxidační-číslo) "-itý")
        ((= 4 oxidační-číslo) "-ičitý")
        ((= 5 oxidační-číslo) "-ičný, -ečný")
        ((= 6 oxidační-číslo) "-ový")
        ((= 7 oxidační-číslo) "-istý")
        ((= 8 oxidační-číslo) "-ičelý")))
```

Několik myšlenek o logice

```
(define pravda #t)
(define (je-pravda? co)
  (and pravda co))

(define (neplatí-že co)
  (if (je-pravda? co)
      (not pravda)
      pravda))

(define (nepravda? co)
  (neplatí-že (je-pravda? co)))

; Implikace, viz https://cs.wikipedia.org/wiki/Implikace
(define (když první-věta druhá-věta)
  (cond
    ((and (nepravda? první-věta) (nepravda? druhá-věta))
     pravda)
    ((and (nepravda? první-věta) (je-pravda? druhá-věta))
     pravda)
    ((and (je-pravda? první-věta) (nepravda? druhá-věta))
     (neplatí-že pravda))
    ((and (je-pravda? první-věta) (je-pravda? druhá-věta))
     pravda)))
```


uzávěry

iterace

abstrakce

cyklus

rekurze

deklarativní

programování

imperativní

algoritmus

anonymní

funkce

vyššího řádu

```
; Když prší, je mokro.
(define prší pravda)
(define je-mokro pravda)

; Možné negace věty 'Když prší, je mokro.'
(define když-neprší-je-mokro
        (když (neplatí-že prší) je-mokro))

(define když-prší-není-mokro
        (když prší (neplatí-že je-mokro)))

(define když-neprší-není-mokro
        (když (neplatí-že prší) (neplatí-že je-mokro)))

; Jaká je negace věty 'Když prší, je mokro.'?
(equal? (neplatí-že (když prší je-mokro))
        když-neprší-je-mokro)

(equal? (neplatí-že (když prší je-mokro))
        když-prší-není-mokro)

(equal? (neplatí-že (když prší je-mokro))
        když-neprší-není-mokro)
```

Osm kouzelných jazyků a jednoduchý problém

Kouzelných (programovacích) jazyků je hodně (moc velká spousta). Pro zajímavost je tu řešení jednoduchého problému v osmi různých kouzelných jazycích.

Řešení problému v různých kouzelných jazycích obsahují příklady toho, jak pojmenovat podstatná jména, jak funguje rozhodování a jak tvořit nová kouzelná slovesa. Příklady řešení slouží jako ukázky skladby vět různých kouzelných jazyků a v tuhle chvíli není důležité jim rozumět, jen se na chvilku potrápit nad možným významem.

> Podstatné jméno (číslo) 20 je pojmenované jako dvacet a nové kouzelné sloveso větší-než pracuje s jedním podstatným jménem (parametrem) číslo: jestli je číslo větší než dvacet, kouzelné sloveso větší-než vrátí nové podstatné jméno (text) "Větší je číslo.", kde číslo v textu je nahrazené konkrétní aktuální hodnotou; jinak vrátí "Větší je 20.".
>
> Pro otestování kouzelného slovesa větší-než se použijí podstatná jména (čísla, argumenty) 11 a 27.

Racket

Řešení v kouzelném jazyce Racket (nebo Scheme) se dá spustit pomocí kouzelné hůlky DrRacket.

```
; Je větší 20 nebo číslo?
(define dvacet 20)
(define (větší-než číslo)
  (if (> číslo dvacet)
    (string-append "Větší je " (number->string číslo) ".")
    (string-append "Větší je " (number->string dvacet) ".")))
(větší-než 11)
(větší-než 27)
```


Prolog

Kouzelné jazyky jsou univerzální nebo specializované, na některé věci se hodí více, na některé méně. Třeba na logiku je skvělý kouzelný jazyk Prolog.

```
% Je větší 20 nebo číslo?
dvacet(D) :- D is 20.
vetsiNez(X) :- dvacet(D), X > D, format("Větší je ~d.", X).
vetsiNez(X) :- dvacet(D), X < D, format("Větší je ~d.", D).

vetsiNez(11).
vetsiNez(27).
```

Je třeba si uvědomit, že příklady řešení jsou skutečně jen *příklady řešení* - jednoduchý problém lze ve stejném kouzelném jazyce vyřešit více způsoby, stejně jako třeba v češtině jde jednu informaci říci různě.

Forth

Historie kouzelných jazyků je pestrá a každý kouzelný jazyk je v něčem jedinečný. Stejně tak i čarodějové jsou různí a každý čaroděj má své oblíbené kouzelné jazyky. Třeba kouzelný jazyk Forth.

```
\ Je větší 20 nebo číslo?
: dvacet 20 ;
: větší-než
    dup dvacet >
    if
    else drop dvacet
    then ." Větší je " . ." ." ;
11 větší-než
27 větší-než
```

Uiua

Kouzelný jazyk Uiua je velmi kompaktní. Nejen takové kouzelné jazyky existují.

```
# Je větší 20 nebo číslo?
Dvacet   ← 20
VětšíNež ← $"Větší je _." ⍥(Dvacet⍟)(∘) > Dvacet .
VětšíNež 11
VětšíNež 27
```


JavaScript

Kouzelnou hůlkou pro kouzelný jazyk JavaScript je webový prohlížeč. Třeba ve Firefox stačí zmáčknout F12, přepnout na záložku "Konzole" a vést dialog s kouzelnou hůlkou. Většinou jsou ale věty kouzelného jazyka JavaScript součástí webových stránek.

```
// Je větší 20 nebo číslo?
var dvacet = 20;
function větší_než(číslo) {
  if (číslo > dvacet)
    return "Větší je " + číslo + ".";
  else
    return "Větší je " + dvacet + ".";
}
větší_než(11);
větší_než(27);
```

Python

Kouzelný jazyk Python vede v průzkumech o nejpopulárnější a nejjednodušší programovací jazyky.

```
# Je větší 20 nebo číslo?
dvacet = 20
def větší_než(číslo):
    if číslo > dvacet:
        print(f"Větší je {číslo}.")
    else:
        print(f"Větší je {dvacet}.")

větší_než(11)
větší_než(27)
```

C

Některé kouzelné hůlky umí jen přečíst dopis a nelze s nimi vést dialog. Třeba kouzelný jazyk C je případ kouzelného jazyka, pro který existuje spoustu (hodně moc) kouzelných hůlek, ale jen s málokterou se dá vést dialog. Kouzelný jazyk C je *legendární* a je v něm napsáno hodně operačních systémů - programů, které jsou potřeba, aby se zařízení jako počítač dalo vůbec spustit.

```c
/* Je větší 20 nebo číslo? */
#include <stdio.h>
#include <stdlib.h>

int const dvacet = 20;

char *
vetsi_nez(int const cislo)
{
    char *str = (char *)malloc(100 * sizeof(char));
    if (NULL == str) {
        exit(1);
    }
    if (cislo > dvacet) {
        sprintf(str, "Větší je %d.", cislo);
    } else {
        sprintf(str, "Větší je %d.", dvacet);
    }
    return str;
}
```



```c
int
main()
{
    char *str;

    str = vetsi_nez(11);
    printf("%s\n", str);

    str = vetsi_nez(27);
    printf("%s\n", str);

    free(str);
    str = NULL;
    return 0;
}
```

Shell

Podobně *legendární* je i kouzelný jazyk Shell. Shellů je spoustu (hodně) a mezi nejznámější patří POSIX Shell a Bash. Shell souvisí s příkazovou řádkou operačního systému - když v Linux zapneme příkazovou řádku, ve většině případů tak začínáme s počítačem komunikovat v kouzelném jazyce Bash.

```shell
#!/bin/sh
dvacet=20
vetsi_nez()
{
        if [ $1 -gt $dvacet ]
        then
                echo "Větší je $1."
        else
                echo "Větší je $dvacet."
        fi
}
vetsi_nez 11
vetsi_nez 27
```