

# IB111 Úvod do programování skrze Python

## Přednáška 12

Vývoj software

Nikola Beneš

11. prosinec 2015

# Osnova dnešní přednášky

- povrchní náhled na návrh software
  - příklad návrhu jednoduchého programu
- dokumentace
- styl psaní programů
- testování
- ladění programů
  
- bonus: správa verzí

## Příklad návrhu programu

Podle 9. kapitoly knihy John M. Zelle:  
Python Programming: An Introduction to Computer Science

*„Proč ve valné většině prohraju, i když jsem jen o trochu horší?“*

## Racquetball

- americký sport, podobný squashi
- body se získávají jen při podání
  - prohraje-li podávající hráč, ztrácí podání
  - (podobné jako stará pravidla volejbalu)
- vyhraje, kdo získá 15 bodů

## Simulace hry

- chceme simulovat racquetballový zápas
- hráči se zadanou úspěšností



**Vstup:** program se zeptá na potřebné údaje

- pravděpodobnost výhry hráče A při podání
- pravděpodobnost výhry hráče B při podání
- počet her

**Výstup:** výsledné statistiky

- počet her
- počet výher pro hráče A (v procentech)
- počet výher pro hráče B (v procentech)

*Poznámka:* Nebudeme kontrolovat správnost zadaných údajů.

## Top-down design

- systematický přístup k návrhu
- začínáme na vysoké úrovni abstrakce
- řešení problému vyjádříme pomocí menších problémů
- pokračujeme, dokud jsme problém nerozbili na problémy úplně triviální
- pak to všechno naprogramujeme

## Základní algoritmus

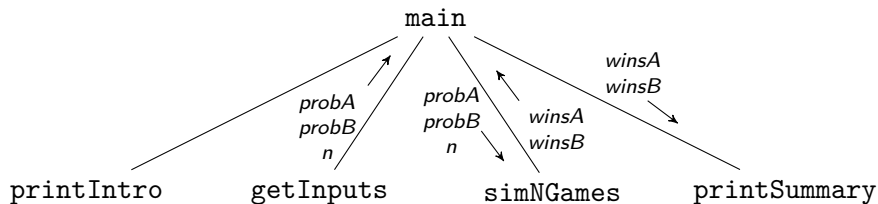
- vypiš úvodní informace
- získej vstupní údaje:  $probA$ ,  $probB$ ,  $n$
- simuluj  $n$  her racquetballu s použitím  $probA$  a  $probB$
- vypiš kolik her se hrálo, kolikrát vyhráli jednotliví hráči

*Jaká data si budou jednotlivé kroky mezi sebou předávat?*

## Princip oddělení zodpovědnosti (Separation of Concerns)

- čtyři nezávislé úkoly
- jasně specifikované rozhraní (vstupy/výstupy)
- funkce `main` se nemusí starat o to, *jak* funguje funkce `simNGames`, zajímá ji jen, *co* spočítá
- abstrakce umožňuje ignorovat ostatní detaily

# Návrh shora dolů – první úroveň





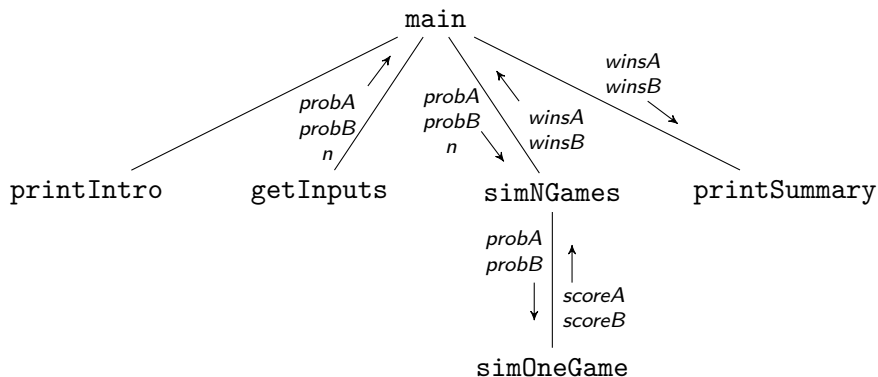
## Druhá úroveň návrhu

- opakujeme stejný postup pro zbývající části
- funkce:
  - `printIntro`
  - `getInputs`
  - `simNGames`
  - `printSummary`
- tři z těchto funkcí můžeme napsat přímo

## Funkce `simNGames`

- nastav `winsA` a `winsB` na nulu
- opakuj `n`krát:
  - simuluj jednu hru
  - pokud vyhrál A
    - zvyš `winsA` o jedničku
  - jinak
    - zvyš `winsB` o jedničku
- nakonec vrať `winsA` a `winsB`

## Návrh shora dolů – druhá úroveň



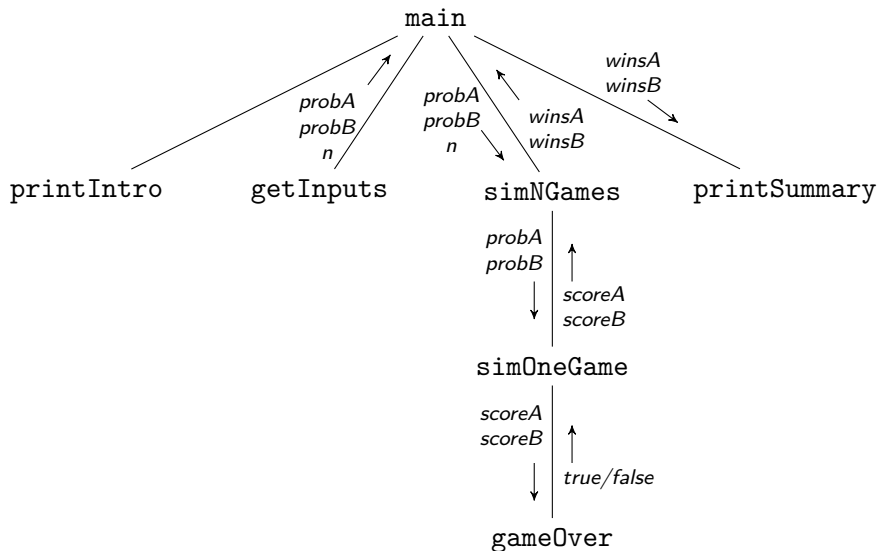
## Třetí úroveň návrhu

- pokračujeme dál
- funkce `simOneGame`

## Náčrt algoritmu

- nastav skóre obou hráčů na nulu
- nastav podávajícího hráče na A
- dokud hra neskončila:
  - simuluj jedno podání aktuálně podávajícího hráče
  - aktualizuj skóre
- vrať skóre obou hráčů

## Návrh shora dolů – třetí úroveň



## Poslední kroky

- funkce `gameOver`
- a náš program je hotov!

## Shrnutí

1. vyjádřili jsme algoritmus jako sadu menších problémů
2. ujasnili jsme si rozhraní každého menšího problému (vstup/výstup)
3. popsali jsme algoritmus s využitím rozhraní menších problémů
4. tento proces jsme opakovali pro každý menší problém

## Programy obsahují chyby

- chyby jsou běžné
- všichni chybují

## Testování programu

- po částech (*unit testing*)
- půjdeme zdola nahoru
  - nejprve otestujeme funkci `gameOver`
  - máme-li důvěru v to, že `gameOver` funguje správně, půjdeme dále
- jak testovat systematicky?

## Přístupy k návrhu

- viděli jsme: návrh shora dolů (top-down design)
- jiné přístupy: prototypování a spirálový vývoj

## Prototypování

- prototyp: jednoduchá verze programu, ne zcela funkční
- návrh, implementace, testování prototypu
  - vhodné pro ověření, zda to, co děláme, má vůbec smysl
- prototyp dále vylepšujeme (inkrementální vývoj)
- časem získáme kompletní program



## Fáze prototypování

1. První prototyp: Hra 30 podání, podávající má vždy 50% šanci výhry. Pomocný výpis po každém podání.
2. Dva parametry pro pravděpodobnost výher.
3. Test na konec hry (15 bodů). Máme funkční simulaci jedné hry.
4. Opakování několika her. Výstup: počet vyhraných her.
5. Kompletní program. Interaktivní vstup, pěkně formátovaný výstup.

## Umění návrhu

- existuje řada různých návrhových technik
  - shora dolů
  - prototypování
  - objektově-orientovaný návrh
  - ...
- mohou se vzájemně doplňovat
- není žádný *nejlepší způsob návrhu*
- návrh software je kreativní proces

## Testování, ladění, dokumentace, styl

# O čem přemýšlet při vytváření programů

## Základní otázky

- je můj program dobře napsaný?
  - budu mu rozumět, až jej uvidím po půl roce?
  - bude mu rozumět někdo jiný?
- funguje můj program správně?
  - dává očekávané odpovědi?
  - má očekávané chování?
  - co je vlastně očekávané?
- funguje můj program efektivně?
  - je dostatečně rychlý?
  - nevyužívá příliš mnoho paměti?
  - a co jiné zdroje (síťová komunikace, ...)?

## Dokumentace

- píšeme pro sebe i pro ostatní
- komentáře v kódu
- dokumentace rozhraní
- názvy (modulů, funkcí, proměnných)

## Dokumentace v Pythonu

- dokumentační řetězec (*docstring*)
- první řetězec funkce (třídy, metody, modulu)

[ukázka]

*Nejlepší kód je takový, který se dokumentuje sám.*

## **Styl psaní programů** (coding style)

- programovat se dá pěkně i škaredě
- různá doporučení
  - závisí na jazyce, na konkrétní společnosti apod.
  - nejlépe je být konzistentní

## **Rozumná doporučení pro Python**

- <https://www.python.org/dev/peps/pep-0008/>
- odsazení
- rozumná délka řádku
- bílé místo (mezery mezi operátory)
- pojmenování proměnných (funkcí, tříd, metod, modulů, ...)
- a další ...

# Testování a ladění programů

(rozsáhlé téma, zde jen velmi lehce nakousneme)

## Testování

- velmi důležitá součást vývoje software
- různé úrovně testování
  - od jednoduchých testů nejmenších kusů kódu (unit testing) až po testování celého rozsáhlého systému
- automatické testování
  - kód, který má za cíl otestovat jiný kód
  - různé nástroje
- mnoho vstupů (snaha pokrýt co nejvíc možností)

## Porovnání efektivity různých programů

- měření času (spotřebované paměti, jiných zdrojů)
- mnoho vstupů
- jiný způsob?

# Testování a ladění programů

## Neúplnost testování

- programy mohou mít mnoho různých vstupů (často nekonečno)
- čím víc vstupů otestuji, tím víc programu věřím
- ale: nikdy nemám úplnou jistotu

## Verifikace

- aktuální výzkumné téma: jak ověřovat správnost programů?
- existují nástroje pro verifikaci
- úplné automatické ověřování programů není možné

## Jiné možnosti

- použití lidského mozku: důkaz správnosti programu
- o správnosti a efektivitě by měl programátor přemýšlet už při návrhu



## Ladění programů

- už jste viděli debugger v prostředí IDLE
- podobných nástrojů existuje celá řada (pro různé jazyky)
- typická funkcionality:
  - výpis aktuálních obsahů proměnných
  - výpis funkcí na zásobníku
  - krokování funkcí
  - breakpoints (určená místa přerušení v programu)
- jiné možnosti:
  - pomocné výpisy
  - logování

# Správa verzí

**Ztráta dat** je většinou nepříjemná.

**Vracení se ke starým verzím** se občas hodí.

**Spolupráce:** co když jeden program vyvíjí více lidí?

**Správa verzí** (version control)

- hodí se nejen při programování
  - webové stránky
  - dokumenty
  - apod.
- uchovává jednotlivé verze souborů
- umožňuje spolupráci vývojářů
  - co když dva pracují na stejném souboru?
- větvení
  - experimentální verze vs. stabilní verze

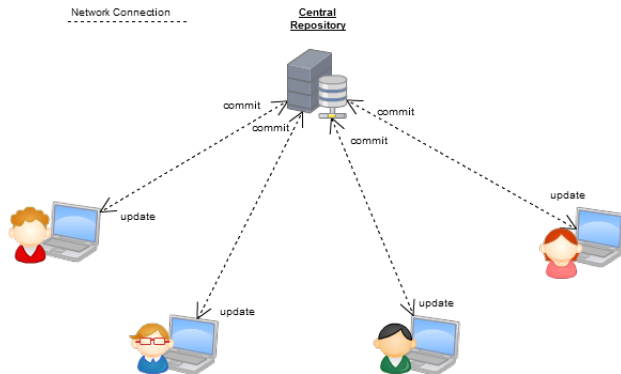
## Centralizovaná

- repozitář na jednom místě (server)
- typický zástupce: Subversion (SVN)

## Distribuovaná

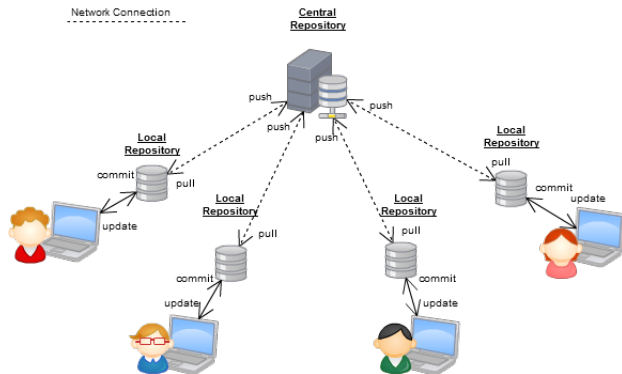
- každý má svůj repozitář
- dá se používat i centralizovaně
  - kopie centrálního repozitáře
- velmi populární: Git, Darcs, Mercurial, Bazaar, ...
- podpora v podobě webových služeb: GitHub, BitBucket, ...

## Centralizovaná správa verzí



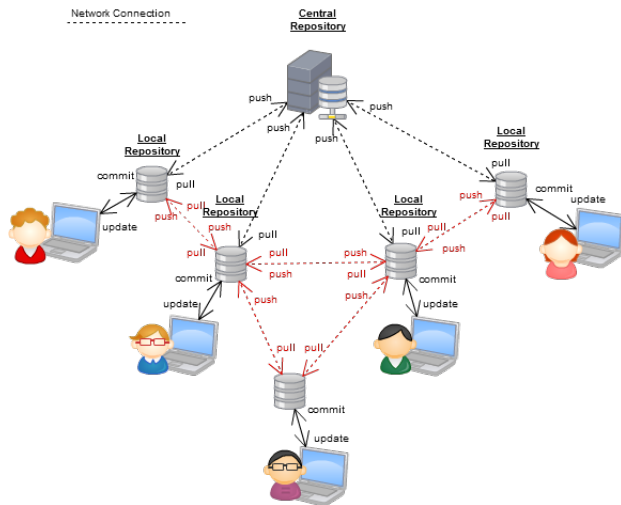
zdroj: <http://programmers.stackexchange.com/a/35080>

## Distribuovaná správa verzí (typické použití)



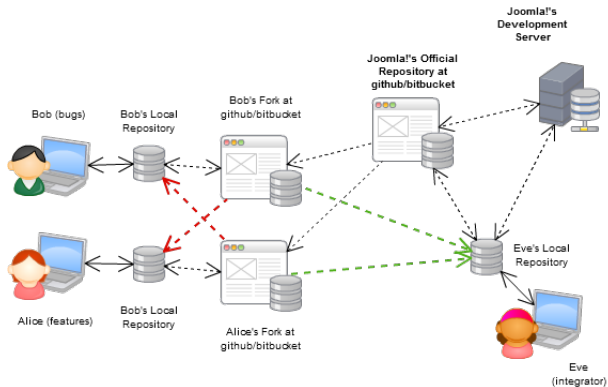
zdroj: <http://programmers.stackexchange.com/a/35080>

## Distribuovaná správa verzí (plné použití)



zdroj: <http://programmers.stackexchange.com/a/35080>

## Distribuovaná správa verzí (použití v open-source projektech)



zdroj: <http://programmers.stackexchange.com/a/35080>



## Užitečné odkazy (SVN)

- fakultní administrativa: <https://fadmin.fi.muni.cz/auth/>  
(založení Subversion účtu)
- SVN: <http://subversion.apache.org/>
- TortoiseSVN: <https://tortoisesvn.net/>  
(SVN klient pro Windows)

## Užitečné odkazy (Git)

- oficiální stránka: <https://git-scm.com/>
- GitHub: <https://github.com/>
- vyzkoušejte si GitHub: <https://try.github.io>

## Návrh a vývoj software

- kreativní proces
- existují nějaké doporučené techniky návrhu

## Dokumentace, styl psaní programů

- dodržujte rozumné konvence (buďte konzistentní)
- používejte komentáře, dokumentujte funkce apod. (v rozumné míře)

## Testování

- velmi užitečné, ale
- nikdy nezaručí úplnou správnost

## Ladění programů (debugging)

## Správa verzí

- spolupráce
- vracení se ke starým verzím
- centralizovaná (SVN) vs. distribuovaná (Git, ...)