Lekce 1
(Teoretická část)

Webové programování (server, klient, php, html, css, js)

Adresa, aplikační routování, vstupní bod aplikace

- Server má nastavené základní routování: pro danou ip adresu a port odkáže na root adresářové struktury
- Další specifikace adresy přes lomítka funguje jako určení složky stejně jako ve Windows či na serveru v konzoli (linux bash)
- Většinou je pro přístup zakázáno vše kromě veřejné složky a i zde je většinou téměř vše kromě indexu zakázané pro přímý přístup (htaccess) => je tedy docela dobře možné, že třeba /index.php je jediný přímo přístupný soubor
- Server je nakonfigurován tak, aby se v případě určitých souborů určitým způsobem zachoval
    - soubory se obecně vrací tak, jak jsou (text, obrázky, binarky, **také html**) bez dalšího zpracovávání
    - soubory php ale server začne zpracovávat a začne se řídit jeho příkazy (= kód) (v dovolené míře)
- Vnitřní routování si již tradičně řídí aplikace sama, tedy logika odvýjející se z indexu.
    - Veškerý obsah je obyčejně v neveřejné části serveru, ale přesto je zasílán v response klientovi, což je zprostředkováno právě aplikační logikou kódu php.
    - = Jsme plně v rukou aplikační logiky php kódu a tato logika rozhoduje o tom, co se nám v response (= odpovědi serveru na “request”) vrátí.

Php vs HTML, CSS, JS

- Jediné php je zpracováváno serverem. Zbytek (html, css, obrázky, binarky, ale i js) je bez jakékoliv interpretace zasláno klientovi, který se sám rozhodne, jak s obsahem naložit.
- Tradičním klientem je webový prohlížeč. Prohlížeč obdrží response jako plaintext, nic víc. Sám je tím, kdo tento strukturovaný plaintext přeloží do audiovizuální (obrázky, html, css) či logické (js) podoby.
- Podobně jako server zpracovává php a nabízí k dispozici zdroje serveru, klient zpracovává JavaScript a nabízí zdroje PC/prohlížeče

Přesah k desktopovemu programování

- Javascript běží v prohlížeči, tedy si v něm lze programovat také pro pouze lokální účely.

- To samé platí pro HTML/CSS (což ale není imperativni, turingovsky kompletní programování)
- Zpracování php lze také zprovoznit na lokálním stroji (server, PC a další výpočetní stroje se od sebe v zásadě neodlišují) a programovat si tak pro lokální potřeby.
- Jako jsou pro server soubory .php určené k imperativnímu zpracování, tak ve Windows to jsou např. exe (kompilovaný) či .bat. (interpretovaný). Za ekvivalent prohlížeče zde lze považovat Windows okno (explorer)

Kódování, programování, softwarová architektura

- Kódování je schopnost převádění myšleného obsahu/objektivně komunikovatelné ideje do zapsatelné (kódovatelné) podoby. ⇒ zápis not, zápis barev, zápis chutí, zápis slov a vět, stejně jako zápis algoritmů = vše je kódování (umožňuje dekódování bez nutnosti předání inteligentní bytostí)
- Programování lze od (softwarového) kódování odlišit tím, že zde se nejedná o zápis něčeho hotového, ale o vymýšlení algoritmickeho řešení pro daný problém.
- Softwarová architektura se od prostého programování liší tím, že lépe zvažuje celkové fungování systému se zvážením budoucího vývoje.

Dvě paradigmata programování (deklarativní a imperativní)

- K pojmům “deklarativní” a “imperativní” se uchyluji spíš z nutnosti. Dávám jim význam, který jim možná správně nenáleží
    - ⇒ chci jimi vyjádřit dva podstatně odlišné způsoby přemýšlení při programování, kdy tyto způsoby nejsou volbou programátora, ale jsou mu vnucovány daným programovacím jazykem (v širokém smyslu, kdy za *programovací jazyk* označuji také třeba HTML)
- Reálné programování (např v jazyce php, C#, python...) je dle mého vysvětlení vždy kombinací deklarativního a imperativního přístupu. Oproti tomu např. psaní HTML, CSS kódu či SQL dotazování je pouze deklarativní.

<u>Imperativní paradigma a turingovsky kompletní programovací jazyky</u>

- Jde o paradigma, ve kterém jsou **chronologicky a za sebe** (spíše **pod sebe**) řazeny **příkazy** (= imperatum) výpočetnímu stroji.
- Základem je uvažování o “krokovátku”, které říká, v jakém kroku, u jakého příkazu, se v danou chvíli nacházíme a co bude následovat.
- Např.: php či C#, přičemž krokovátko lze přímo sledovat při spuštění nástroje zvaného “debugger”

- Pojem “turingovsky kompletní” je v důsledku daleko komplikovanější, ale v zásadě jde o minimální nutné podmínky pro možnost naprogramovat a provést libovolný logický algoritmus.
- Algoritmus bude zapsán v jednotlivých krocích, příkazech.
- Protože všechny moderní běžné programovací jazyky jsou turingovsky kompletní, jsou si zároveň navzájem v tomto smyslu ekvivalentní. Všemi jazyky mohu vykonat všechny algoritmy. Není tomu tedy tak, že některým z těchto jazyků můžeme vykonávat pouze určité algoritmy a jiným jazykem zase jiné (kladivem nevyvrtám díru a vrtákem zase těžko budu zatloukat hřebíky).
- To neznamená, že jsou všechny jazyky stejné. Některým z jazyků se určitý problém může řešit snadněji/příhodněji než jiným.
- Základními vlastnostmi turingovsky kompletního programovací jazyka jsou **zhruba** tyto schopnosti:
    - Na vyžádání příkazem zapisovat do paměti na určité místo a číst z paměti z určitého místa (je jedno, zda jde o HDD, SSD, RAM, magnetonovou pásku či blok papíru)
    - Postupovat v jednotlivých příkazech.
    - Vyhodnotit logickou podmínku “if” a dle jejího vyhodnocení vykonat tu či jinou sadu příkazů. Základem je vykonání příkazu “goto”, kdy se krokovátko plnění příkazů přesune na jiné místo a odtud pokračuje.
    - Vyhodnotit logické operace jako rovnost, nerovnost, větší, menší a provést základní početní úkony: minus, plus, násobení, dělení.
    - ⇒ pokud daný stroj a jeho jazyk tyto věci umožňuje, můžeme již zapsat víceméně libovolný algoritmus a jsme již omezování pouze časem, pamětí a poruchovostí stroje

<u>Deklarativní paradigma</u>

- Spíše negativní vymezení vůči imperativnímu paradigmatu: Jde o paradigma, ve kterém kus kódu zapsaný v určité posloupnosti neodráží nijak chronologii zpracování. Jednotlivé kusy kódu **nejsou příkazy, které by nějaké “krokovátko” postupné zpracovávalo**. Výpočetní stroj takto zapsaný kód uchopí jako **celek**, který teprve následně zpracovává svou logikou, která však nijak nemusí odpovídat posloupnosti zápisu. Výpočetnímu stroji nepřikazujeme, co má v jaké posloupnosti vykonat, ale deklarujeme, co si přejeme za výsledek.
- Příkladem je jazyk SQL.
    - Příkazem:
      Select kniha
      From knihovna
      Where titulek = ‘Lolita’;
        - Neříkáme, že má stroj nejprve vybrat knihu. Na prvním řádku vůbec nevíme, o jakou knihu by se mělo jednat. Příkaz “from knihovna” také

nedává smysl sám o sobě atd. Např. první řádek tedy prostě nelze sám o sobě vykonat, to je nesmysl.
    - Deklaraci lze tedy chápat jako příkaz, který ale dává smysl pouze jako celek.
    - Podobně fungují lidské věty. Věta “Šel jsem spát” nedává moc smysl po jednotlivých slovech, ale jako celek. “Šel” - kdo? Kam? “Jsem” - samo smysl nedává ve spojení s “šel” už lepší - asi jsem někam šel. “Spát” ...

Propojení obou přístupů v praxi programování i samotných programovacích jazycích

- Běžné moderní programovací jazyky je dle mého správné chápat jako kombinaci obou paradigmat: imperativního i deklarativního.
- Imperativní paradigma je sice základem (viz turingovsky kompletní jazyky), ale modernější jazyky přidávají stále více deklarativních prvků.
- Např. zcela základní deklarace funkcí. Funkce může být definována nahoře či dole v kódu a je to úplně jedno. Když program funkci volá, tak najednou ví, kam si pro ni sáhnout. Funkce je vnitřně imperativní (= tzv. Implementace funkce), ale z vnějšího pohledu je deklarativní (říká o sobě, jaké přijímá argumenty a jaké hodnoty vrací a tím je plně deklarována).
- Stejně tak deklarace tříd v objektovém programování je v zásadě deklarativní. Základnější jazyky jako např. C++ dokonce tyto deklarace oddělují od zbytku kódu. (Také třeba PL/SQL)

Známé a rozšířené programovací jazyky

- Java, C# (vysoká abstrakce, staticky typované, kompilované)
- Python, PHP (vysoká abstrakce, volně typované, interpretované)
- C++ (střední stupeň abstrakce, staticky typovaný, kompilovaný)
- Assembler (nízký stupeň abstrakce, ???, ???)
- Fortran (historický, nízký stupeň abstrakce, ???, ???)
- Mnoho dalších…

Podstata (imperativního) programovacího jazyka, knihovny jádrové a další rozšíření

- O imperativních programovacích jazycích jsme řekli, že jsou navzájem ekvivalentní ve smyslu jejich potenciálu vyjádřit jimi bez zbytku libovolný logicky konzistentní algoritmus (turingovská kompletnost). Zároveň jsme řekli, že ale nejsou identické.
- ⇒ Čím se tedy odlišují? Co tvoří podstatu každého z nich?
    - **Reserved words ~ keywords**
        - = (zpravidla latinkou psaná) slova, která nemohou být použita jako názvy identifikátorů (např.: název proměnné), a která tvoří základní stavební

prvky pro vytvoření skladby smysluplné v rámci daného programovacího jazyka.
    - ***[V analogii ke skutečnému jazyku se jedná o slova, která zná slovník daného jazyku. Například slovo "horse" je slovem jazyka angličtina ale nikoli čeština. "Gift" je slovem užívaným jak v angličtině, tak v němčině, ale v každém má jiný význam. "Robot" je slovo existující v češtině i angličtině a má zároveň shodný význam v obou.]***
    - Oproti lidským hovorových jazykům obsahuje slovník programovacího jazyka pouze hrstku výrazů. Stačí jich cca do 50 pro vyjádření většiny situací.
- **Syntax (tedy skladba)**
    - Gramatika jazyka. Určuje, jaké skladby jsou v daném jazyce platné, i když ne nutně smysluplné.
    - Jako je v češtině věta "Jdu lesem." platná a "Lesem červený :::: lesem" nikoli, tak některé skladby jsou v daném jazyce platné a jiné nikoli.
    - V kompilovaných jazycích (C#) je chybná skladba zachycena při procesu kompilace (= **compile error**). V případě interpretovaných jazyků (php) dojde k odhalení pochybení ve skladbě až při běhu programu (= **runtime error**)
- ⇒ některé odlišnosti v kombinacích keywords-syntax mezi jazyky jsou povrchní a stačí nahradit jedno slovo za jiné či mírně upravit skladbu. Jiné odlišnosti jsou zásadnějšího rázu, kdy je v každém z jazyků třeba přemýšlet radikálněji odlišným způsobem.
    - Povrchní odlišnosti:
        - Např.: If-else skladba:
          C#, php: if (podminka) {} else if (podminka) {}
          PL/SQL: if (podminka) then … elsif (podminka) then … end if;
            - ***[Podobně v lidských jazycích: Jana snědla jablko. Jane ate an apple.]***
    - Zásadnější odlišnosti:
        - Pole hodnot:
          C#: pole vždy deklaruje svůj typ: např.: string[] a do pole nemůže přijít nic jiného než string
          Php: pole žádný typ nedeklaruje, je prostě polem hodnot či objektů a mohou se libovolně mísit: Array().
        - C#: v polích nemá žádné vlastní klíče, vždy se jedná o kontinuální indexy.
        - Php: umožňuje definovat klíčet, např.: Array("fruit" => "banana");
            - ***[Podobně v lidských jazycích: Jdu parkem. I walk in the park.]***

Jádrové knihovny (knihovny funkcí/metod)

- Každý programovací jazyk má zároveň svou jádrovou knihovnu funkcí, která sice netvoří samotnou podstatu programovacího jazyka, ale běžně je možné ji považovat za jeho součást.
- V php například běžně můžeme použít funkci: in_array(). Php jazyk si však lze představit i bez ní a zda je nějaká hodnota v poli či nikoli můžeme zjistit vlastní metodou tak, že budeme iterovat polem a porovnávat hodnoty. Na přítomnost funkce in_array() se však můžeme spolehnout kdekoli máme něco do činění s jazykem php.

Další knihovny (knihovny funkcí/metod)

- V php například známá knihovna fpdf, která umožní konstruovat PDF soubory. Tato knihovna rozhodně netvoří podstatu jazyka PHP a pokud bychom chtěli, tak si můžeme shodnou knihovnu napsat v php také sami.
- Knihoven je mnoho a některé se postupně stávají standardem. Některé knihovny mohou být na úrovni malweru (jedná se o kusy kódu, které s důvěrou spouštíme). Některé jsou otevřené, jiné nikoli (zejména v kompilovaných jazycích). Mají odlišné licence použití.