14. HTML

HyperText Markup Language je značkovací jazyk pro tvorbu webových stránek a aplikací. Byl vytvořen pro World Wide Web. V roce 1989 Tim Berners-Lee a Robert Caillau vytvářeli propojení informačních systémů pro CERN. V té době se užíval převážně PostScripta SGML, ale z práce Tima Berners-Lea v roce 1990 vznikla nová služba, která změnila pohled na Internet, byl to právě World Wide Web. Tim Berners-Lee v tomto roce navrhl HTML, HTTP protokol a spustil první webový server na světě. Tim je také tvůrcem prvního prohlížeče na světě WorldWideWeb, který byl pouze textový. První prohlížeč s grafickým uživatelským prostředím se jmenuje Mosaic.

HTML dokument je tvořen **tagy** a jejich atributy. Jsou 2 druhy tagů **Párové** a **Nepárové**.

```
<p align = „center“>< /p>
<br />
```

Tagy by se neměly křížit!

Vývoj HTML

- 1991 – 1993 → verze 0.9 – 1.2; bez podpory grafiky
- 1995 → verze 2.0; interaktivní formuláře a podpora grafiky
- 1997 → verze 3.2; tabulky, stylové prvky, W3C validátor
- 1997 → verze 4.0; podpora rámců (<frame>) další prvky formuláře; měla být finální verze HTML, budoucí náhradník = XHTML
- 2014 → verze 5.0; aktuální verze, přidány nové tagy, modernizace a příprava pro multimediální obsah
  - Nový tag pro multimédia <audio>, <video>
  - Nový tag pro doctype <!DOCTYPE html>
  - Přidány tagy pro hlavičku a patičku <header>, <footer> …

Validace

Zpětná kontrola HTML kódu a detekce syntaktických chyb. Oficiální validátor je od firmy W3C.

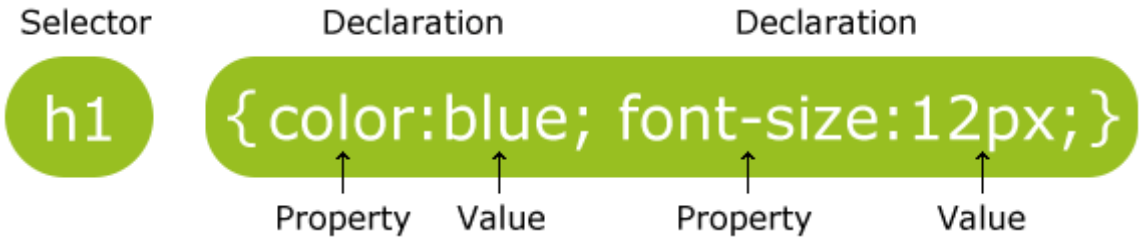
Při tvorbě webových stránek, či HTML5 aplikací se nepoužívá pouze samotné HTML, ale mnoho dalších jazyků. Nejhlavnější z nich jsou: JavaScript, CSS, XML, AJAX. Na straně serveru: PHP, ASP…

15. CSS

Cascading Style Sheets. Kaskádové styly. Jazyk pro popis zobrazení jednotlivých elementů v dokumentu HTML (XHTML). Vznikly v roce **1997** jako oddělení grafické části HTML od samotného HTML souboru. Aktuální verzí CSS je verze CSS3, která přinesla novinky ohledně HTML5 a je zcela kompatibilní se starými CSS specifikacemi. CSS spravuje W3C (stejná firma, která spravuje i HTML).

Kaskádový styl je stylopis, množina pravidel, která se používá jako grafická nástavba klasického HTML. HTML neumožňuje tak rozmanitě nastavovat vzhled stránky a prvků na stránce.

Možnosti zápisu



Inline CSS

```
<h1 style="color: red">Tento nadpis bude červený.</h1>
```

Stylopis

Do hlavičky se napíše stylopis mezi tagy <style></style>.

```
<head>
    <title>Test</title>
    <style>
        p {color: red}
    </style>
</head>
```

Externí CSS soubor

Soubor s příponou **.css**

Do hlavičky se přidá:

```
<link rel="stylesheet" type="text/css" href="./test.css" />
```

CSS soubor bude vypadat následovně:

```
p { color: red }
h1 {tag: hodnota}
```

Selektory

CSS selektory se používají k výběru (hledání) HTML elementu na základě jejich jména, ID, třídy (class)…

Element selector

Vybírá element na základě jejich jména.

Pokud mají být všechny elementy stejně nastylované stačí pouze:

```
p {
    text-align: center;
    color: red;
}
```

ID selector

Využívá ID atribut HTML elementu ke specifikování určitého elementu.

ID by mělo být unikátní (pro celou stránku), aby přesně identifikoval element.

ID by nemělo začínat číslem.

Identifikátor lze přiřadit pouze k jednomu elementu.

HTML:

```
<h1 id="para1"> TEST </h1>
```

CSS:

```
#para1 {
    text-align: center;
    color: red;
}
```

Class selector

"Třídní Selektor" vybírá elementy podle specifického **class** atributu.

HTML:

```
<h1 class="center"> TEST </h1>
```

CSS:

```
.center {
    text-align: center;
    color: red;
}
```

Popřípadě specifikace/oddělení jednotlivých elementů.

```
p.center {
    text-align: center;
    color: red;
}
```

Sjednocování selektorů

Pokud existuje vice elementů se stejnými styly, lze je sjednotit.

```
h1 {
    text-align: center;
    color: red;
}
h2 {
    text-align: center;
    color: red;
}
p {
    text-align: center;
    color: red;
}
```

Místo dlouhého a zbytečného zápisu lze použít následující zkrácený zápis:

```
h1, h2, p {
    text-align: center;
    color: red;
}
```

Pseudotřídy

Speciální události.

U odkazů lze nastavit, jak bude vypadat, když se přes něj přejede myší…

```
a:link {color: #FFE400; text-decoration: none;}
a:visited {color: #FFE400; text-decoration: none;}
a:active {color: #FF9000; text-decoration: none;}
a:hover {color: #FF9000; text-decoration: none;}
```

Pozicování

Pomocí CSS lze nastylovat, jak budou jednotlivé elementy umístěny.

Existují 2 základními druhy pozicování:

Relativní

Určuje pouze, o kolik se má objekt posunout oproti své normální pozici

Absolutní

Umístí objekt do stránky bez ohledu na okolní text

Fixní

Speciální pozice, která udává, že objekt zůstává na stejném místě při rolování.

16. Práce se soubory v programovacích jazycích

Soubor

Pojmenovaná sada dat uložených na paměťovém médiu, se kterou lze pracovat jako s jedním celkem.

V programování slouží k uložení dat z programu nebo načtení dat do programu. Data ze souboru a do souboru putují přes streamy (proudy).

K souborům lze přistupovat dvěma způsoby a to **textově** nebo **binárně**. Rozdíl je v tom, jak se data čtou (zapisují).

Práce se soubory v jazyce JAVA (viz. PRM 17)

Pro práci se soubory v jazyce JAVA slouží třída **File**. Zároveň slouží jako manažer souborů. Na začátku programu je nutné importovat knihovnu **java.io.File**.

Popřípadě **java.io.FileWriter**, **java.io.FileReader,** a jiné knihovny, které jsou potřeba. Popřípadě rovnou importovat celou input/output knihovnu **java.io***.

Aby bylo možné pracovat se soubory (textovými či binárními), musí se vytvořit **Streamy** (proudy). Stream je proud dat z programu na nějaké místo na disku, konkrétně k souboru, s nímž pracuji.

Stream může být vstupní pro čtení, nebo výstupní pro zápis, existuje i varianta, že lze číst a zapisovat do souboru pomocí jednoho proudu.

Streamy lze dál dělit dle toho, jaká data v něm proudí. Bajtové, znakové, datové, standardní, objektové a proudy vyrovnávací paměti.

Po dokončení práce se soubory, by se měly uzavřít všechny proudy, které byly otevřené. Hlavně výstupní proud. Pokud se neuzavře výstupní proud, data se neuloží.

Třídy pro práci se soubory

Základní třídy pro práci s textovými soubory jako jsou ***.txt**, ***.java**, ***.sql** a mnoho dalších jsou **java.io.FileReader** pro čtení a **java.io.FileWriter** pro zápis.

Obou třídám v konstruktoru lze předat buď objekt třídy File, nebo řetězec s cestou k souboru.

Při vytváření instance třídy FileReader může nastat vžjimka **java.io.FileNotFoundException** pokud soubor neexistuje. Pokud soubor neexistuje a vytváří se instanci třídy FileWriter, tak se soubor vytvoří, ale může nastat **java.io.IOException** pokud soubor nelze vytvořit.

Práce se soubory v jazyce PHP

V jazyce PHP k vytvoření souboru slouží metoda fopen:

```
$soubor = fopen("soubor.txt", "w+");
```

Místo w+ lze použít další hodnoty:

- **a** – otevře soubor pro přidání, pokud soubor neexistuje, PHP jej vytvoří
- **a+** - otevře soubor pro přidávání a čtení, pokud soubor neexistuje, PHP jej vytvoří
- **r** – otevře soubor pouze pro čtení.
- **r+** - otevře soubor pro čtení a zápis
- **w** – otevře soubor pro zápis, původní data budou ztracena. Pokud soubor neexistuje, PHP jej vytvoří
- **w+** - otevře soubor pro zápis a čtení, původní data budou ztracena, pokud soubor neexistuje, PHP jej vytvoří
- **x** – vyhodí error, pokud soubor existuje. Nemusí se existence souboru ověřovat pomocí **file_exists()**;
- **c** – zamyká soubor. Nemusí se použít flock, aby se zamkl soubor a další request nemohl zapisovat do stejného souboru.

Zavření souboru se provádí stejnou funkcí jako vytvoření. Po dokončení práce se souborem je rozumné jej zavřít.

```
fclose(id_souboru);
fclose($soubor); //fopen vrací ID souboru
```

Práce se soubory v jazyce VBA

V jazyce VBA slouží pro práci se soubory funkce:

```
Open nazev_souboru For rezim As #cislo
```

Režimy pro práci se soubory:

- Input
  - Jednosměrný vstup ze souboru do programu (čtení dat ze souboru)
- Output
  - Jednosměrný výstup z programu do souboru (zápis dat do souboru přepsáním, pokud již existuje soubor se stejným jménem)
- Append
  - Jednosměrný výstup z programu do souboru (zápis konec - přidávání na konec)
- Random
  - Náhodný přístup, čtení i zápis
- Binary
  - Binární režim pro čtení i zápis

Příkaz CLOSE

```
Close #cislo
```

- Uzavírá (ukončuje přístup) otevřený soubor s uvedeným číslem
- Bez uvedení čísla uzavírá všechny otevřené soubory v programu
- Dokončuje poslední zápisy z vyrovnávací paměti – vyprázdní buffer
- Přesunuje celý soubor z operační paměti na trvalé paměťové médium

Funkce EOF

```
Eof (cislo)
```

- EOF = End Of File (konec souboru)
- Funkce vrací logickou hodnotu
  - True – zda bylo dosaženo konce souboru
  - False – zda nebylo dosaženo konce souboru

Příkaz Input

```
Input #cislo, seznam
```

- Seznam identifikátorů jednoduchých proměnných (nikoli polí)
- Jednotlivé identifikátory jsou odděleny čárkou
- Data ve čteném souboru musí svým charakterem a polohou odpovídat typu jednotlivých proměnných v seznamu

Příkaz LINE INPUT

```
Line Input #cislo, promenna
```

- Přečte ze souboru jeden řádek zakončený znakem s kódem 13 (Cariege Return – Konec řádku, Návrat vozíku)
- Čteny jsou všechny znaky v daném řádku (včetně mezer, čísel, uvozovek a jiných znaků)
- Znaky konce řádku – Chr(13) a Chr(10) – jsou z řetězce vypuštěny
- Další příkaz přečte následující řádek

Příkaz GET

```
Get #cislo, zaznam, promenna
```

- Obdobně jako LINE INPUT přečte jeden řádek (záznam), ale lze specifikovat libovolný záznam (řádek v databázi)
- Cislo – manipulační číslo otevřeného souboru
- Zaznam – pořadové číslo záznamu (řádku), který se má přečíst

Příkaz PRINT

```
Print #cislo, seznam
```

- Zapisuje data do otevřeného souboru
- Oddělovače v seznamu
  - ; – znak se zapíše na další pozici
  - , – znak se napíše do další zóny (jedna tisková zóna je 14 znaků)
- Každý příkaz PRINT bude psát na nový řádek
- Oddělovač použitý na konci seznamu způsobí, že další příkaz PRINT bude zapisovat na stejný řádek (potlačení přechodu na další řádek)

Příkaz WRITE

```
Write #cislo, seznam
```

- Výrazy jsou v seznamu odděleny čárkou
- Každý příkaz WRITE píše na nový řádek
- Řetězcové hodnoty jsou ve výstupním souboru ohraničeny uvozovkami
- Jednotlivé hodnoty jsou ve výstupním souboru odděleny čárkou

Příkaz PUT

```
PUT #cislo, zaznam, promenna
```

- Výstup do souboru

Příkazy GET a PUT jsou vhodné pro jednoduchou obsluhu databází, neboť dovolují manipulovat se záznamy v libovolném pořadí na rozdíl od sekvenčního přístupu.

Práce s adresáři

```
ChDir cesta 'nastaví pracovní cestu
ChDrive disk 'změna aktuální jednoty
MkDir cesta 'vytvoří adresář
```

17. Datová struktura pole

Skupina proměnných obvykle stejného typu, které jsou v paměti alokovány za sebou. K těmto proměnným lze přistupovat pomocí jejich indexů (pořadí v této posloupnosti, počítáno od 0). Z tohoto způsobu alokace je zřejmé, že pole má fixní délku, kterou nelze nijak změnit (protože paměť v oblasti za polem může být obsazena jiným objektem).

Dělení Polí

- **Statické**
  - Velikost je jistá před překladem, interpretací programu, obvykle indexované od 0
- **Dynamické**
  - Velikost závisí na průběhu programu, a lze měnit i po jeho vytvoření
  - Pošle se požadavek na správce paměti a ten buď přidělí paměť, nebo oznámí chybu
  - V Javě nelze měnit velikost pole za běhu programu
- **Indexované**
  - Přístup k prvku pomocí čísla, většinou od 0
  - Java má přístup k prvkům pouze pomocí indexu
- **Asociativní**
  - Přístup pomocí řetězce
  - V PHP lze přistupovat pomocí řetězce
- **Homogenní**
  - V poli může být jen jeden datový typ
- **Různorodé**
  - V poli můžou být různé datové typy
  - V Javě pole nemůže být různorodé
  - V PHP pole může být různorodé

Vícerozměrná pole

Vícerozměrná pole v Javě jsou ve skutečnosti jen pole plné polí.

Při deklaraci vícerozměrných polí se postupuje obdobně jako u pole jednorozměrného, jediným rozdílem je počet závorek, který se uvádí.

Dvojrozměrné pole je časté, trojrozměrné je neobvyklé a více jak trojrozměrné je obvykle chyba návrhu aplikace (jenom málokdo si umí taková data představit).

```
Typ[][] jmeno = new typ[pocet_prvku][pocet_prvku];
```

JAVA

V okamžiku, kdy se pole vytvoří, tak jsou jeho hodnoty přednastaveny do výchozího stavu. V případě integerů 0, u booleanu false, a v případě referencí ukazatel do prázdna null.

```
typ[] jmeno;
```

Vytvoří se proměnná pole, ale zatím se nealokuje místo v paměti.
Místo v paměti se alokuje konstruktorem.

```
jmeno = new typ[velikost_pole];
```

Inicializaci a deklaraci pole lze zapsat obdobně jako u ostatních typů.

```
typ[] jmeno = {hodnota, hodnota2, hodnota3, hodnota4};
typ[] jmeno = new typ[velikost pole];
```

PHP

V PHP existuje spoustu syntaxí pro vytvoření pole.

```
$pole = array("test", 2, 2.16);
$pole1 = array(0 => "test", 1 => 2, 2 => 2.16);
$pole2 = array("pozice" => "test", "pozice2" => 2, "pozice3" => 3.14);
$pole3 = ["test", 2, 2.16];
$pole4 = [0 => "test", 1 => 2, 2 => 2.16];
$pole5 = ["pozice" => "test", "pozice2" => 2, "pozice3" => 3.14];
```

Jak je vidět na příkladech, pole v PHP je **asociativní** a **různorodé**.
V PHP lze vytvořit i vícerozměrné pole.

```
$a1 = array("a" => 0, "b" => 1);
$a2 = array("aa" => 00, "bb" => 11);
$together = array($a1, $a2);
```

VBA

```
Dim nazev(velikost) As Datovy_typ
```

Metody nad třídou Array:

Třída Array obsahuje pomocné metody pro práci s poli.

- **Sort**
  - Metoda pro třídění pole
- **Reverse**
  - Obrácení posloupnosti
- **IndexOf, LastIndexOf**
  - Tyto metody vrátí index prvního nebo posledního nalezeného prvku
- **Copy**
  - Zkopíruje část pole do jiného pole
  - Prvním parametrem je zdrojové pole, druhým cílové a třetím počet znaků, který se má zkopírovat

Metody nad polem:

Třída Array není jedinou možností, jak s polem manipulovat. Přímo na samotné instanci pole (konkrétní proměnné) lze volat také spoustu metod.

- **Length**
  - Vrátí délku pole
- **Min, max, average, sum**
  - Vracejí minimum, maximum, průměr, součet všech prvků
  - Bez parametrů
- **Concat, intersect, union**
  - Vrátí na výstupu nové pole
  - Jako parametr mají druhé pole
  - Spojení; Průnik
  - Sjednocení; prvky, které byly v obou, jsou zde pouze jednou
- **First, last**
  - Vrátí první, poslední prvek
- **Take, skip**
  - Berou jako parametr počet prvků
  - Take vrátí pole s daným počtem prvků zkopírovaných od začátku
  - Skip naopak vrátí pole bez těchto prvních prvků
- **Contains**
  - Vrací True/False podle toho, zda se prvek v parametru metody, nachází v poli
- **Reverse**
  - Vrátí nově vytvořené pole s otočenými prvky
- **Distinct**
  - Zajistí, aby byl v poli každý prvek jen jednou
  - Vrátí nově vytvořené pole

18. Vyhledávání

Způsob zjištění, zda je hledaný prvek přítomen v prohledávaných datech. Vyhledávání probíhá v organizované datové struktuře (pole, spojový seznam, arraylist, soubor…). Vyhledávací algoritmy typicky vrací logickou hodnotu (**true** / **false**), která indikuje, zda byl hledaný prvek nalezen. Nejčastěji prohledáváme pole.

Mezi parametry patří **složitost**. S ní související hardwarové nároky, nároky na vstupní data (struktura, datové typy, nutnost seřazení prvků).

Možnosti algoritmu jsou dány jazykem, ve kterém je programován. To ovlivňuje například datové struktury a typy, které mohou být prohledávány (neobjektové jazyky neumožní použití spojových seznamů, striktně typované jazyky neumožňují použití nehomogenní datové struktury).

Vyhledávací algoritmy

Brute Force (Naivní vyhledávání)

Algoritmus prochází celé pole, od prvního prvku až po poslední. U každého prvku kontroluje, zda není shodný s hledaným. I když najde shodu, prochází celé pole do konce.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury
- Není třeba upravovat pole
- Jednoduchou úpravou lze získat další funkce

Nevýhody

- Prohledává celé pole

Složitost = N

```
public static boolean bruteForce(int[] array, int search){
        int count = 0;
        boolean result = false;
        for(int i = 0; i<array.length; i++){
            if(array[i] == search){
                result = true;
                count++; //počítání výskutů
            }
        }
        System.out.println(count);
        return result;
}
```

Vyhledávání bez zarážky

Algoritmus prochází pole, dokud nenajde hledaný prvek. Když najde shodu, pole dál neprochází.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury
- Není třeba upravovat pole

Nevýhody

- Dva testy v každém kroku (zda nebyl nalezen prvek a zda není konec pole)

Složitost = 2*N

```
public static boolean withoutStop(int[] array, int search){
        boolean result = false;
        int i = 0;
        while((i < array.length) && (result == false)){
            if(array[i] == search){
                result = true;
            }
            i++;
        }
        return result;
}
```

Vyhledávání se zarážkou

Aby byla vyloučena nutnost dvou testů bez rizika, že se „sáhne“ mimo pole, přidá se na konec pole další prvek, který bude stejný jako hledaný.

Pokud se přidává další prvek jako **zarážka** je nutné pole překopírovat do nového a většího.

Poté stačí jediný test pro každý krok, protože je zaručené, že prve bude vždy nalezen a vyhledávání skončí.

Nakonec je nutné zjistit, zda byl hledaný prvek v původním poli. Pokud byl hledaný prvek nalezen na posledním indexu, znamená to, že v původním poli nebyl.

Výhody

- Jednoduchost
- Aplikovatelný pro všechny datové typy a struktury

Nevýhody

- Nutná úprava pole

Složitost = N+1

```
public static boolean withStop(int[] array, int search){
        int newArray[] = new int[array.length+1];
        System.arraycopy(array,0,newArray,0,array.length);
        newArray[array.length] = search;
        boolean result = false;
        int i = 0;
        while (result == false){
            if(newArray[i] == search){
                result = true;
            }
            i++;
        }
        if(i == newArray.length){
            return false;
        }else{
            return true;
        }
}
```

Binární vyhledávání

Nejdříve se otestuje prvek uprostřed. Je-li nalezena shoda, vyhledávání skončí.

Pokud ne, porovná se velikost a podle ní se rozhodne, ve které polovině pole se bude dále vyhledávat.

Z vybrané poloviny se vybere prostřední prvek a vyhledávání pokračuje až do doby, kdy je prvek nalezen, nebo než už pole dále nejde dělit.

Zde se vytvoří dva indexy do pole, **levý**, který ukazuje na první prvek pole, a **pravý**, který ukazuje na poslední.

Z těchto se poté počítá prostřední prvek a tento prvek se porovná s hledaným prvkem.

Pokud se nerovnají, pak se buď levý index posune doprava, nebo pravý doleva. Toto se opakuje, dokud se nepřekříží.

Výhody

- Efektivnost

Nevýhody

- Nutná úprava pole (seřazení)
- Pouze pro číselné datové typy

Složitost = $\log_2(N)$

```
public static boolean binary(int[] array, int search){
    int left = 0;
    int right = array.length - 1;
    do{
        int middle = (left + right)/2;
        if(array[middle] == search){
            return true;
        }else if(array[middle] > search){
            right = middle - 1;
        }else{
            left = middle + 1;
        }
    }while(left <= right);
    return false;
}
```

19. Řadící algoritmy I (Bubble, Select, Insert)

Řadící algoritmy zajišťující seřazení daného souboru dat do specifikovaného pořadí. Nejčastěji se řadí podle velikosti čísel nebo abecedně.

Bubble Sort

V poli se porovnávají 2 sousední prvky.

Pokud je číslo vlevo větší (menší) než vpravo, prohodí se.

Po průchodu celým polem je seřazený minimálně 1 prvek. Pole se o seřazený prvek zkrátí a pokračuje znova, dokud pole není seřazeno.

Počet porovnání = $(n^2 - n) / 2$

Počet přesunů = přibližně n^2

Složitost = n^2

```
public static void bubble(int[] array){
        for (int i = 0; i < array.length-1; i++){
            for (int j = 0; j < array.length-1-i; j++){
                //if(array[j] < array[j+1]){ směr řazení
                if(array[j] > array[j+1]){
                    int tmp = array[j];
                    array[j] = array[j+1];
                    array[j+1] = tmp;
                }
            }
        }
}
```

Select Sort

Najdeme maximum (minimum) v poli.
Tento prvek se prohodí s prvkem na posledním (prvním) místě.
Postup se opakuje a po každém průchodu bude vynechán poslední (první) prvek.
Počet porovnání = $(n^2 - n) / 2$
Počet přesunů = přibližně n-1
Složitost = n^2

```
public static void select(int[] array){
        for (int i = 0; i < array.length-1; i++){
            int max = pole.length-i-1;
            for (int j = 0; j < array.length-i; j++){
                //if(array[j] < array[max]){ směr řazení
                if(array[j] > array[max]){
                    max = j;
                }
            }
            int tmp = array[array.length-1-i];
            array[array.length-1-i] = array[max];
            array[max] = tmp;
        }
}
```

Insert Sort

Postupně prochází prvky a každý nesetříděný se zařadí na správné místo.

K tomu je nutné mít pole již částečně setříděné. To se provede tak, že se první prvek považuje za setříděný a začne se od druhého.

Počet porovnání = $(n^2 - n) / 2$

Počet přesunů = přibližně $(n^2 – n - 2) / 2$

Složitost = přibližně n^2

```
public static void insert(int[] array){
     for (int i = 0; i < array.length; i++){
          int tmp = array[i];
          int j = i - 1;
          //while((j>=0) && (array[j] < tmp)){směř řazení
          while((j >= 0) && (array[j] > tmp)){
               array[j+1] = array[j];
               j--;
          }
          array[j+1] = tmp;
     }
}
```

20. Řadící algoritmy II (Quick - sort)

Jeden z nejrychlejších běžných algoritmů řazení porovnávání prvků. Vymyslel jej **Sir Charles Antony Richard Hoare** v roce 1962.

Quick Sort

Rozdělí řazené posloupnosti čísel na dvě přibližně stejné části.
V jedné části jsou čísla větší a ve druhé menší, než nějaká zvolená hodnota (pivot).
Pokud je tato hodnota zvolena dobře, jsou obě části přibližně stejně velké.
Pokud budou obě části samostatně seřazeny, je seřazené i celé pole.
Obě části se pak rekurzivně řadí stejným postupem (Lze zapsat i iteračně).
Složitost = n^2 (nejhorší možný případ)

```
public static void quick(int[] array, int left, int right){
    int i = left;
    int j = right;
    int pivot = array[(left+right)/2];
    do{
        //while(array[i] > pivot) i++;
        //while(array[j] < pivot) j--;
        while(array[i] < pivot) i++;
        while(array[j] > pivot) j--;
        if(i <= j){
            int tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
            i++;
            j--;
        }
    }while(i<j);
    if((j-left) > 0){
        quick(array, left, j);
    }
    if((right-i) > 0){
        quick(array, i, right);
    }
}
```

Lomuto Quick Sort

Nico Lomuto v roce 1984 přišel s vylepšením Quick Sortu. Principem je postupné umisťování prvků menších jak pivot doleva. Levá půlka pole se zvětšuje. Výhodou je méně porovnání, ale nevýhodou je více přesunů než Hoareho metoda.

Složitost závisí na zvolení vhodného pivota, může degradovat na N^2.

21. MySQL – DDL

DDL

Příkazy DDL (**Data Definition Language**) slouží pro definici dat.

Umožňují:

- Přidávat
- Upravovat
- Mazat logické struktury, které obsahují data (databáze, tabulky, indexy, pohledy,….)

CREATE

Slouží pro vytvoření databázových objektů. Všechny jeho možnosti syntaxe se mohou lišit podle typu databáze.

```
CREATE DATABASE `knihovna` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE TABLE kniha(
    id INTEGER NOT NULL AUTO_INCREMENT,
    nazev VARCHAR(50) NOT NULL,
    autor VARCHAR(50) NOT NULL,
    nakladatelstvi VARCHAR(50) NOT NULL,
    rok_vydani YEAR(4) NULL,
    isbn VARCHAR(13) NOT NULL,
    PRIMARY KEY(id),
    UNIQUE KEY(isbn)
) ENGINE = InnoDB DEFAULT CHARSET utf8;
```

ALTER

Slouží ke změně databázových objektů.

Přidání sloupce

```
ALTER TABLE kniha ADD COLUMN k_sehnani SET ('ano', 'ne', 'mozna') NOT NULL;
```

Změna nastavení sloupce

```
ALTER TABLE kniha CHANGE COLUMN k_sehnani je_na_sklade ENUM('ano', 'ne') NOT NULL;
```

Odstranění sloupce

```
ALTER TABLE kniha DROP COLUMN je_na_sklade;
```

DROP

Slouží k odstranění databázových objektů.

```
DROP DATABASE 'knihovna';
DROP TABLE `kniha`;
```

Datový typ	Délka	Rozsah
INT, INTEGER	4	-2 147 483 648 až 2 147 483 647
FLOAT		-3.402823466E+38 až 3.402823466E+38
DOUBLE		-1.7976931348623157E+308 až 1.7976931348623157E+308
DATE	‘0000-00-00’	"RRRR-MM-DD"; 1000-01-01 až 9999-12-31
DATETIME	‘0000-00-00 00:00:00’	"RRRR-MM-DD HH:MM:SS"); 1000-01-01 00:00:00 až 9999-12-31 23:59:59
TIME	‘00:00:00‘	"HH:MM:SS"; "-838:59:59" až "838:59:59"
YEAR(n)	0000	
CHAR(n)		Pevná délka; 0-255
VARCHAR(n)		0-255
BLOB, TEXT		maximálně 65 535 znaků
LONGBLOB, LONGTEXT		maximálně 4 294 967 295 znaků
ENUM('item1','item2',...)		pole předem definovaných řetězců o maximálním počtu 65 535

22. MySQL – DML

DML

Příkazy DML(**Data Manipulation Language**) slouží pro získání dat z databáze a pro jejich úpravy.

INSERT

Přidá do tabulky nový záznam.

Vložení více záznamů na jednou

```
INSERT INTO kniha (id, nazev, autor, nakl, pocet, rok, isbn, precteno)
VALUES
    (NULL, 'Cosi', 'Kdosi', 'Kdesi', 500, 1978, '0-13-110163-3', 'ano'),
    (NULL, 'Nekdo', 'Ona', 'Kdesi', 460, 1998, '0-13-659723-8', 'ano');
```

INSERT IGNORE INTO

Při pokusu o vložení řádku shodujícího se v primárním či unikátním klíči potlačí standardní chování databázového stroje (hození chybové hlášky a skončení jako celku), místo toho bude tuto chybu ignorovat a pokračovat dál. Má význam zejména v případě vkládání více záznamů – po skončení příkazu budou vloženy jen ty nekolidující řádky, ty kolidující — duplicitní vloženy nebudou.

UPDATE

Upravuje data (záznamy) v relační databázi. Může být upraven jediný záznam, nebo i více záznamů najednou. Upravené záznamy musí odpovídat definované podmínce.

```
UPDATE kniha SET pocet_stran = 236 WHERE id = 1;
```

DELETE

Slouží k odstranění záznamů z tabulky.

WHERE je volitelný parametr, bez udání smaže všechny záznamy v tabulce, ale ne samotnou tabulku.

```
DELETE FROM kniha WHERE id = 1;
```

23. MySQL – SQL

SQL

Patří do skupiny příkazů pro manipulaci s daty (**DML**). Vybírá data z databáze a umožňuje výběr podmnožiny a řazení dat.

```
SELECT * FROM cosi;
```

Specifikace

WHERE

Slouží k filtraci dat. Uvede se podmínka, pokud je splněna, vyberou se data

```
SELECT * FROM cosi WHERE id=1;
```

LIKE

Oproti WHERE nabízí použití „patternů“, zástupných znaků.

JOIN

Pokud se vybírají data z více tabulek.

ORDER BY

Seřazení dat. Specifikace sloupce (podle, kterého se má řadit) a směr řazení ASC, DESC

```
SELECT nazev, zanr, rezie FROM cosi WHERE rezie = 'Menzel' ORDER BY nazev DESC;
```

GROUP BY

Seskupení dat. Nejběžnější použití je získání počtu záznamů odpovídající každé jednotlivé hodnotě jiného sloupce, časté je také získání součtu, aritmetického průměru či jiných statistických hodnot z vybíraných záznamů

```
SELECT zanr, COUNT(*) FROM filmy GROUP BY zanr;
```

DISTINCT

Nevypisuje duplicitní záznamy

Agregační funkce

COUNT – počet
SUM – součet
MIN – minimum
MAX – maximum
AVG – průměr

24. PHP – Formuláře a jejich zpracování

Formuláře

Důležitá součást stránek, slouží k přihlašování k některým stránkám (eshopy, bakaláři, diskuze na webu apod.), k odeslání údajů do databáze nebo k vyplnění anketních otázek. Samo o sobě formuláře nejsou součástí PHP, ale už HTML jazyka.

Formulář je definován párovým tagem `<form> ... </form>`. Formuláře se používají k odesílání dat na server, který je zpracuje. Jedna HTML stránka může obsahovat žádný, jeden, nebo libovolně mnoho formulářů na jedné stránce Důležité atributy tagu <form> jsou **ACTION** (skript, který zpracovává data, určuje, kam se budou data posílat, není-li uvedeno, odešlou se data téže stránce) a **METHOD** - určuje jakým způsobem se budou data posílat.

Způsoby posílání dat

GET

Data se budou předávat jako součást URL.

POST

Data jsou součástí dotazu.

Vhodné pro posílání hesel a obsáhlejších dat…

Vstupní pole

Pro zadávání hodnot od uživatele se používá nepárový tag <input />. Input jako takový má spoustu atributů pro různé typy vstupů. Předávají se mu určité důležité atributy:

- Type – Specifikuje, jak bude input vypadat.
- Name – Jméno, kterým se dané pole identifikuje
- Value – Výchozí hodnota, která je předvyplněná

Text

Pro zadávání textových vstupů od uživatele.

Password

Pro zadávání textových vstupů uživatele, které nebudou čitelné.

Checkbox

Slouží pro výběr. Výběr N hodnot z M. Pokud není checkbox zatrhnut, neodesílá se.

Radio

Slouží pro vybrání 1 možnosti z N možností; Checked

Select

Slouží pro vybrání 1 možnosti z N možností pomocí rolovací nabídky; Nepárový tag option (value) mezi tagama se udá, co má uživatel vidět.

Submit

Slouží k odeslání dat vyplněných ve formuláři. Atribut value → Nápis, který bude na tlačítku.
Odesílá se vždy, pokud má atribut name.

Superglobální proměnné

PHP používá zvláštní pole, ve kterých jsou uložená data.

- **$GLOBALS –** všechny běžně používané proměnné v PHP (i všechny superglobální pole; $GLOBALS včetně)
- **$_SERVER –** různé systémové proměnné, údaje o serveru, uživateli…
- **$_ENV –** proměnné poskytované skriptu z prostředí
- **$_POST –** proměnné poskytované skriptu přes **HTTP POST**
- **$_GET –** proměnné poskytované skriptu přes **HTTP GET**
- **$_COOKIE –** proměnné poskytované skriptu přes **HTTP COOKIES**
- **$_FILES –** proměnné poskytované skriptu přes HTTP POST upload souborů
- **$_REQUEST –** proměnné poskytované skriptu přes libovolný vstupní mechanismus (POST, GET…)
- **$_SESSION –** proměnné registrované pro aktuální session

25. PHP – Cookies a sessions

Sessions

Pojem představující permanentní síťové spojení mezi klientem a serverem, zahrnující výměnu paketů.

Jakmile PHP obdrží příkaz k započetí session, zjistí nejprve, zda již session neběží. Pokud ne vytvoří ji, pokud ano, připojí se k ní. PHP přidělí session identifikátor a vyhradí si někde místo pro ukládání tzv. session-proměnných. Od tohoto místa dále lze u libovolné proměnné zvolit, že bude součástí session, a server si pak její obsah pamatuje mezi stránkami. Session může být kdykoli ukončena. Když se to neprovede, zruší se zavřením prohlížeče.

Pro zapnutí práce se sessionama slouží příkaz `session_start();` a pro zapnutí outputbufferingu `ob_start();`

Pro přístup k sessionám se používá superglobální pole $_SESSION[‘nazev‘];

Pro smazání sessiony lze použít `unset($_SESSION['cosi']);`, `session_unset();`, `session_destroy();`,`$_SESSION = array();`

Cookies

V protokolu HTTP se jako **coookies** označuje malé množství dat, která WWW server pošle prohlížeči, který je uloží na počítači uživatele. Při každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru.

Cookies se odešlou s požadovanou stránkou a to v hlavičce. Ukládají se do souboru na disku (můžou být hashovány). Když se potom prohlížeč připojí na stejný server, bude automaticky posílat cookies.

Na začátku je nutno zadat `ob_start();`. Pak se pracuje stejně jako se sessions, ale nastavení cookie se dělá metodou `SetCookie (nazev_cookie, hodnota ($hodnota), [platnost (time() + 3600)])`. Na závěr aplikace je nutno zadat `ob_end_flush();`.

26. PHP – Propojení s DB

K databázi se připojuje, pokud je třeba pracovat s větším množstvím dat. Databáze je prakticky jediným rozumným řešením. Díky SQL je práce s daty podstatně rychlejší a poskytuje širokou škálu nástrojů pro jejich spravování. Pomocí PHP se poté jen odešle zformulovaný dotaz a získá se již zpracovaný výsledek.

Pokud se jedná o jednodušší projekt, ve kterém bude třeba připojovat se k DB jen v jediném souboru, je možné skript definující parametry připojení (a realizující připojení samotné) vložit přímo do něj. U větších projektů se však vyplatí pro tento účel vytvořit externí skript, který lze pak pokaždé snadno vložit.

Komunikace s DB pomocí myslql_*() je už zastaralou variantou a v novějších verzích PHP plánují vývojáři jejich podporu odstranit. Jejich funkce už v současnosti nahrazuje PDO_MySQL.

```php
<?php
	/**Data source name
	* Typ DB
	* Adresa server
	* Port serveru
	* Jméno DB, ke které se připojuje
	*/
	$dsn	= "mysql:host=127.0.0.1;port=3306;dbname=telefony";
	/**Uživatel, který může spravovat Db*/
	$user	= "root";
	/**Heslo uživatele*/
	$password = "";
	/**Vytvoření instance PDO
	* PDO::ERR_MODE_EXCEPTION → chyby se ohlašují jako výjimky
	* PDO::MYSQL_ATTR_INIT_COMMAND → Kódování
	*/
	try{
		$db = new PDO($dsn, $user, $password,
	[PDO::ATTR_ERRMODE		=> PDO::ERRMODE_EXCEPTION,
	PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8',
	PDO::ATTR_PERSISTENT		=> TRUE]);
	} catch(PDOException $e){
		echo 'Connection failed: ' . $e->getTraceAsString();
	}
?>
```

V souborech, kde je třeba se připojit k DB, tak se pouze vyžádá soubor s připojením.

Důležité funkce pro práci s DB

Prepare

Příprava SQL dotazu. Používají se Placeholdery, za které se poté dosadí přesná data.

Params

Pole parametrů, které se dosazují do dotazu.

Execute

Spuštění dotazu.

Fetch

PDO::FETCH_BOTH, PDO::FETCH_ASSOC…

Vrací pole, které představuje řádek, s parametry.