UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

Katedra technické a informační výchovy

Diplomová Práce

František Peiker

JavaScript jako nástroj pro výuku algoritmizace a programování na 2. stupni ZŠ

Olomouc 2024

vedoucí práce: Mgr. Tomáš Dragon

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a uvedl jsem v ní veškerou literaturu a ostatní informační zdroje, které jsem použil.

V Olomouci dne 17. 4. 2024

vlastnoruční podpis

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Mgr. Tomáši Dragonovi za odborné vedení, za pomoc a cenné rady, které pomohly tuto práci zkompletovat.

Obsah

Úvod

Internet je v dnešní době nedílnou součástí našeho života. Díky neustálému vývoji digitálních technologií je možné se připojit k internetu téměř odkudkoliv na světe. Není tedy divu, že se také rozšiřují technologie pro tvorbu webových stránek. Dříve internet sloužil pouze pro psaní blogů, zveřejňování novinek apod. Vizuální podoba webových stránek byla často neatraktivní v porovnání s moderními přístupy k tvorbě webových stránek. Dnes jsou možnosti internetu takřka neomezené. Streamovací služby jako Netflix, HBO MAX, nebo Disney+ začínají nahrazovat televize, internetové obchody nahrazují kamenné, internetové bankovnictví nahrazuje fyzickou návštěvu banky a sociální sítě pomáhají spojovat lidi napříč velkými vzdálenostmi. Je tedy přirozené, že se technologie pro takto důležitou složku našeho života budou vyvíjet ohromnou rychlostí. Nejrozšířenějším programovacím jazykem pro tvorbu webových stránek je JavaScript, ze kterého vycházejí další technologie jako např. jQuery, Node.js, Angular nebo React.

Společně s rychlým vývojem moderních technologií se modernizuje také školství. Skvělým příkladem takové modernizace je nedávná inovace rámcového vzdělávacího programu na základních a středních školách v oblasti informatického kurikula. Zmíněná inovace se zaměřuje především na rozvoj informatického myšlení a digitální gramotnosti, kde se setkáváme s koncepty jako kódování, šifrování, algoritmizace nebo programování. Na školách se pak například algoritmizace a programování vyučuje pomocí vizuálního programovacího jazyka Scratch. Jakmile žáci vstřebají základní i pokročilé znalosti, není třeba již ve Scratchi pokračovat dál a je možné přejít k některému z plnohodnotných programovacích jazyků. Plnohodnotné programovací jazyky jsou mnohem komplexnější než Scratch a dokáží tak ještě více rozšířit povědomí o algoritmizaci a programování. Přínosem pro žáky je také nahlédnutí do vývojového prostředí, ve kterém se vytvářejí reálné aplikace. Tím si žáci uvědomí, co všechno je třeba udělat, než se jim na obrazovce zobrazí požadovaný obsah. JavaScript je v tomto případě vhodnou volbou. Jeho syntaxe není složitá. Pro pochopení základů JavaScriptu a principů jeho fungování stačí mít elementární znalosti anglického jazyka. JavaScript je komplexní a výkonný nástroj, pomocí kterého se vytvářejí webové aplikace, a dokonce i aplikace pro různé druhy operačních systémů.

Cílem diplomové práce je tvorba ucelené sady metodických návrhů pro učitele informatiky na 2. stupni ZŠ se zaměřením na implementaci jazyka JavaScript do výuky v oblasti algoritmizace a programování. Na konci každého metodického návrhu se bude nacházet samostatná práce pro žáky, kteří si po jejím vypracování upevní učivo a naučí

se přemýšlet nad předloženými problémy logicky a systematicky. Teoretická část se bude mimo jiné zabývat aktuální výukou algoritmizace a programování na základních školách, tvorbou webových stránek a aplikací a problematikou JavaScriptu. V teoretické části se také zaměříme na výukové metody, didaktické zásady a organizační formy, které budou nedílnou součástí metodických návrhů.

V praktické části se budeme věnovat samotné tvorbě metodických návrhů. Ty budou z hlediska obsahu koncipovány od nejjednoduššího učiva po nejsložitější. Pro využití jakéhokoliv metodického návrhu ve výuce bude potřeba, aby měli žáci alespoň základní znalosti v oblasti tvorby webových stránek. První metodické návrhy budou zaměřeny na úvod do jazyka JavaScript. V následujících metodických návrzích budeme JavaScript propojovat s testovacími webovými stránkami, aby žáci viděli v praxi, kde lze různé JavaScriptové příkazy a struktury využít. Posledním metodickým návrhem v sadě bude komplexní projekt obsahující všechny již probrané koncepty z předchozích metodických návrhů.

Diplomová práce může být vhodně využita v pedagogické praxi a sloužit jako výukový materiál v období, kdy se již vyučuje informatika novými moderními přístupy a edukační podklady podobného typu jsou žádoucí.

1 Algoritmizace a programování

Vzhledem k cíli práce, který je zaměřen na implementaci jazyka JavaScript do výuky v oblasti algoritmizace a programování, je nutné tuto oblast nejprve definovat a detailněji přiblížit.

Obvyklá definice programování říká, že jde o vyřešení problému pomocí výpočetní techniky, nejčastěji pomocí stolního počítače, notebooku, či chytrého telefonu. Výstupem programování je potom počítačový program, což je řada instrukcí, které vedou k vyřešení daného problému. Pro vytvoření programu je potřeba si vybrat určitý programovací jazyk. Programovacích jazyků je celá řada, například Python, C#, Java nebo JavaScript. Pomocí zvoleného jazyka pak poskytujeme počítači pokyny. Výsledkem vykonání těchto jednotlivých pokynů ve správném pořadí je právě program. Jedním slovem se přesný sled pokynů pro počítač nazývá algoritmus (Digitální vzdělávání, 2021).

Neformálně lze algoritmus charakterizovat jako posloupnost kroků, které popisují postup při vykonání úlohy. V běžném životě obvykle využíváme neformální a volně definované pojmy, avšak ve vědeckém kontextu je nezbytné, aby tyto pojmy byly založeny na přesně definované terminologii (Brookshear et al., 2013). Formální definice algoritmu by mohla být popsána jako posloupnost nebo řada pravidel, která se používají k manipulaci s omezeným souborem dat s cílem řešit určitou třídu problémů podobného charakteru. Také ho lze dále definovat jako soubor pravidel, který je charakteristický pro specifické výpočty nebo aktivity v oblasti počítačového zpracování (Wróblewski, 2015).

Matematičtí historici identifikovali nejpravděpodobnější původ tohoto termínu: pochází ze jména perského matematika Muhammada ibn Musa al-Chwárizmího, jehož jméno mělo v latině podobu Algorismus. Žil v 9. století našeho letopočtu a proslavil se tím, že vytvořil jasná pravidla, která postupně krok po kroku vysvětlovala zásady operací s desetinnými čísly (Wróblewski, 2015). S tímto pojmem se nejčastěji setkáváme v oblasti vědy a techniky, například v kybernetice nebo programování, kde se algoritmem myslí přesně definovaný postup vedoucí k řešení úlohy.

Každý algoritmus:

- Pracuje se vstupními daty, která pocházejí z přesně definované množiny;
- Generuje specifický výsledek (který nemusí být nutně číselný);
- Je přesně definován (jednotlivé kroky musí být jasně určeny);
- Má konečný počet kroků (musí vždy dojít k výsledku);

- Lze jej využít k řešení celé třídy problémů, nikoliv pouze jednoho konkrétního problému (Wróblewski, 2015).

Můžeme si všimnout, že definice algoritmu požaduje, aby posloupnost jeho kroků byla systematicky uspořádána. To znamená, že jednotlivé kroky algoritmu musí být jasně strukturovány v rámci pořadí jejich provádění. Dalším kritériem pro algoritmus je, že se skládá z vykonatelných kroků, což někteří informatici označují termínem „efektivní“. Když říkáme, že je krok efektivní, máme na mysli, že je proveditelný. Rovněž se po algoritmu vyžaduje, aby obsahoval jednoznačně definované kroky. Tato podmínka znamená, že během provádění algoritmu musí být informace k vykonání jednotlivých kroků jasně a úplně určeny. Jednoduše řečeno, při plnění jednotlivých kroků se nepředpokládá žádná kreativita ze strany člověka nebo stroje, který algoritmus provádí, ale dodržování přesně formulovaných instrukcí. Kromě toho algoritmus musí obsahovat konečný počet kroků, což znamená, že jeho provedení musí vést k dosažení stanoveného cíle (Brookshear et al., 2013).

Při řešení úloh není důležitý pouze správný výsledek, ale také co nejrychlejší a nejefektivnější postup, který za stejných podmínek dojde ke stejnému výsledku. Schopnost tvořit takové algoritmy se nazývá algoritmizace. Je to tedy schopnost vytvářet a formulovat postupy a řešení, které se mohou přenechat k vykonání jinému člověku nebo stroji. Tento člověk nebo stroj by poté měl být schopen bezmyšlenkovitě dospět ke správnému řešení pomocí jednotlivých, za sebou přesně uspořádaných kroků (Otevřená věda, 2018).

Algoritmy musejí být nějakým způsobem reprezentovány, aby byly přínosné také pro jiné osoby nebo stroje, kterým bude algoritmus předán. Lidé si mohou algoritmy předávat pomocí přirozeného jazyka, takže například česky, anglicky nebo obrázkem (skládání nábytku). Přirozený jazyk i obrázky mohou mít často hned několik významů, proto nejsou naprosto spolehlivé a přesné. Nemusí se ale jednat pouze o nepochopení významu, je rovněž možné, že bude algoritmus popsán složitým popisem, nebo naopak příliš jednoduchým, a tak nemusejí být dostatečně popsané všechny potřebné informace k jeho správnému provedení. Jinými slovy, když jazyk není přímo určen k reprezentaci algoritmů, můžeme se setkat s problémy při jeho využívání. Proto se v informatice reprezentují algoritmy pomocí programovacích jazyků, kde je přesně definované, co má každá proměnná, metoda a třída v algoritmu za funkci. Pomocí programovacích jazyků se tak vyhneme dvojznačnosti (Brookshear et al., 2013).

1.1 Výuka algoritmizace a programování na základních školách

V 60. letech 20. století začaly vznikat první rozsáhlé informační systémy, měřené podle rozsahu kódu, který byl potřeba pro vytvoření dané aplikace, převážně v assembleru. I přes to, že programování nylo vnímáno jako činnost závislá na intuici a citu, v procesu vývoje programů se objevovala vážná úskalí. Systémy buď vznikaly rychle, avšak s nízkou spolehlivostí, nebo náklady na vývoj překračovaly původní rozpočet, což zpochybňovalo smysl celého projektu. Chyběly metody a nástroje pro kontrolu přesnosti programů, a vyvinuté aplikace byly testovány, dokud neobsahovaly žádné chyby. Oba tyto faktory – spolehlivost systému a finanční náklady – mají v praxi klíčový význam. Jestliže informační systém banky nefunguje stoprocentně správně, tak by jej banka neměla využívat (Wróblewski, 2015).

V určitém bodě došlo ke zhoršení situace natolik, že se začalo otevřeně hovořit o krizi ve vývoji softwaru. V roce 1968 se konala konference NATO v Německu, kde měli odborníci možnost o této problematice diskutovat. O rok později byla vytvořena zvláštní pracovní skupina v rámci Mezinárodní federace pro zpracování informací (IFIP) s cílem vypracovat metodiku programování (Wróblewski, 2015).

Před několika málo lety většina lidí vnímala programování jako tajemnou dovednost, vyhrazenou pouze odborníkům. Nicméně došlo k zásadním změnám a během krátké doby nás internet, sociální sítě, smartphony a různé aplikace výrazně ovlivnily a změnily náš způsob života k nepoznání. Počítače jsou v dnešní době neodmyslitelnou součástí každodenního života a jsou považovány za samozřejmost. Namísto telefonování se dnes upřednostňují sociální sítě, lidé nakupují přes internet, sledují videa online, hrají hry apod. Avšak nejenom, že lze tyto technologie používat v každodenním životě, lidé se také mohou podílet na jejich vytváření. Pokud se člověk naučí programovat, otevře si tak cestu k vytváření vlastních digitálních uměleckých děl (Vorderman, 2022).

Vše, co počítače provádějí, je ovládáno určitými programy, které někdo musel vytvořit. Ačkoli takové programy mohou připomínat cizí jazyk, jedná se o jazyk, který si každý člověk může rychle osvojit. Mnozí lidé v dnešní době považují programování za jednu z klíčových dovedností. Programování navíc poskytuje praktické dovednosti pro život. Posiluje logické myšlení a schopnost řešit problémy, což je v různých oblastech života, od techniky až po právo, nezbytné (Vorderman, 2022).

Výuka algoritmizace a programování představuje značnou výzvu, protože vyžaduje od žáků osvojení zcela nového přístupu k řešení problémů. Navíc je třeba mít osvojené určité znalosti a kompetence v oblasti logického myšlení, matematiky a matematických struktur,

což může být problematické v nižších ročnících druhého stupně základních škol. Značnou výhodou pro vzdělávání se v oblasti algoritmizace a programování je ovládání anglického jazyka na dobré úrovni, jelikož v mnohých programovacích jazycích vycházejí klíčová slova právě z angličtiny (například if, else, for, break, return, …). Také nelze zavrhnout možnost vyhledat pomoc při programování na různých fórech k tomu určených, jako je Stack Overflow nebo GitHub, kde je drtivá většina dotazů a odpovědí napsaná v anglickém jazyce.

K výuce algoritmizace a programování můžeme přistupovat z teoretického a praktického hlediska. V teoretické části je nezbytné jasně vysvětlit pojem algoritmus, aby žáci pochopili jeho podstatu. Dále je důležité žákům nastínit, jak výpočetní technika tyto algoritmy zpracovává, což umožní žákům systematicky přemýšlet o daných problémech. V praktické části obvykle vzniká problém, když mají žáci sami řešit konkrétní problémové úlohy. Nejprve je třeba žákům poskytnout již vyřešené, nebo jen z části vyřešené úlohy s omezeným rámcem možností, které žáci mohou provést a upravit. Na tyto úlohy je třeba nabalovat další možnosti a funkce postupně tak, aby si na ně žáci pomalu zvykli, a aby byli později schopni nad řešením úlohy přemýšlet kriticky a upravovat jej podle sebe, spíše, než aby slepě opisovali a následovali již hotová řešení (Umíme to, 2024).

Na základních školách se s nástupem nové informatiky začíná vyučovat mimo robotiku a kódování také algoritmizace a programování. Když takové pojmy řeknete žákům základní školy, a ještě k tomu dodáte, že se jimi budou v následujících hodinách zabývat, tak je možné, že se jim informatika znechutí. Proto by bylo moudřejší k takovým žákům, kteří se s touto problematikou setkávají prvně, přistupovat s těmito tématy pomaleji. S učivem zahrnujícím algoritmizaci se žáci mohou v hodinách informatiky setkávat už od třetí nebo čtvrté třídy, tedy od prvního roku, kdy se informatika zahrnuje mezi vyučované předměty. V takovém brzkém věku je vhodné téma předkládat praktickými a interaktivními pomůckami jako jsou například robotické hračky Bee-bot. V tomto věku žáci totiž nejsou, nebo jsou pouze v omezené míře schopni přemýšlet v abstraktní rovině, která je nesmírně důležitá pro rozvoj algoritmického myšlení. Algoritmické myšlení je mimo jiné také kritický prvek pro naučení se programovat.

Pro začátek je třeba žákům vysvětlit co to ten algoritmus vůbec je. Jak již bylo zmíněno, pomocí algoritmů se dostáváme k vyřešení určitých problémů, takže algoritmus můžeme vysvětlit jako sled přesně daných kroků, které nás vždy dostanou k našemu vytyčenému cíli. Žákům se příklad algoritmu může vysvětlit na něčem co znají z každodenního života, jako například oblékání (nejprve si obleču ponožky, až poté se obuji a ne naopak), přechod

přes cestu (rozhlédnu se, pokud něco jede, rozhlédnu se znova, pokud nic nejede, můžu bezpečně přejít) nebo vaření (než zaleji čaj vodou, musím tu vodu nejprve uvařit).

Na druhém stupni základní školy už lze algoritmizaci propojit také s programováním pomocí blokově orientovaného programovacího jazyku Scratch. Scratch je velice příhodný nástroj pro výuku algoritmizace a programování na základních školách díky přehledného uživatelského rozhraní a snadné tvorbě programu, který reprezentují již předem naprogramované bloky kódu. Pomocí nástroje Scratch lze vytvořit spoustu projektů od jednoduchého příběhu až po komplexní hry. Žáci si tak hravou formou vyzkouší skládání jednotlivých bloků za sebe tak, aby se dostali k vytyčenému cíli. Tento nástroj lze tedy zařadit do vyučovacích jednotek kdykoliv napříč výuku algoritmizace a programování.

Nabízí se otázka, jak učitel pozná, nebo kdy rozhodne, že algoritmické myšlení žáků je dostatečné na to, aby se mohli začít učit programovat pomocí plnohodnotného programovacího jazyku. Záleží ovšem na spoustě faktorech, takže se zde nejeví žádná jednoznačná odpověď. Žákům, kteří se podle nové informatiky vzdělávají pouze prvním nebo druhým rokem, se programovací jazyky nejspíš představovat nebudou. Na druhou stranu, žáci mohli mít informatiku jako volitelný předmět, kde už se se všemi potřebnými znalostmi seznámili. V posledních letech školní docházky (v osmé a deváté třídě) by již mělo být možné zakomponovat do výuky informatiky také plnohodnotný programovací jazyk. Většinou si učitelé pro tyto účely vybírají jazyk Python kvůli jeho jednoduché syntaxi a instalaci vývojového prostředí. Kromě programování se na základních školách vyučuje také tvorba webových stránek pomocí jazyků HTML a CSS. Tyto jazyky jsou jistě vhodné pro začátečníky, ale pomocí těchto dvou jazyků žák nedocílí žádné dynamické webové aplikace. Nachází se zde tedy možnost propojení programovacího jazyka s tvorbou webových stránek.

Je logické, že začneme hledat možnost propojení u dalšího programovacího jazyka vhodného pro tvorbu webových aplikací, kterým je JavaScript. Z JavaScriptu vychází knihovna React, spravovaná společností Meta, ze které dále vychází framework React Native. Tento framework si našel obrovskou popularitu v posledních několika letech díky jeho nativnímu přístupu k vývoji aplikací. Díky tomuto frameworku lze totiž vyvíjet jak webové aplikace, tak i aplikace pro operační systémy Android, iOS nebo Windows.

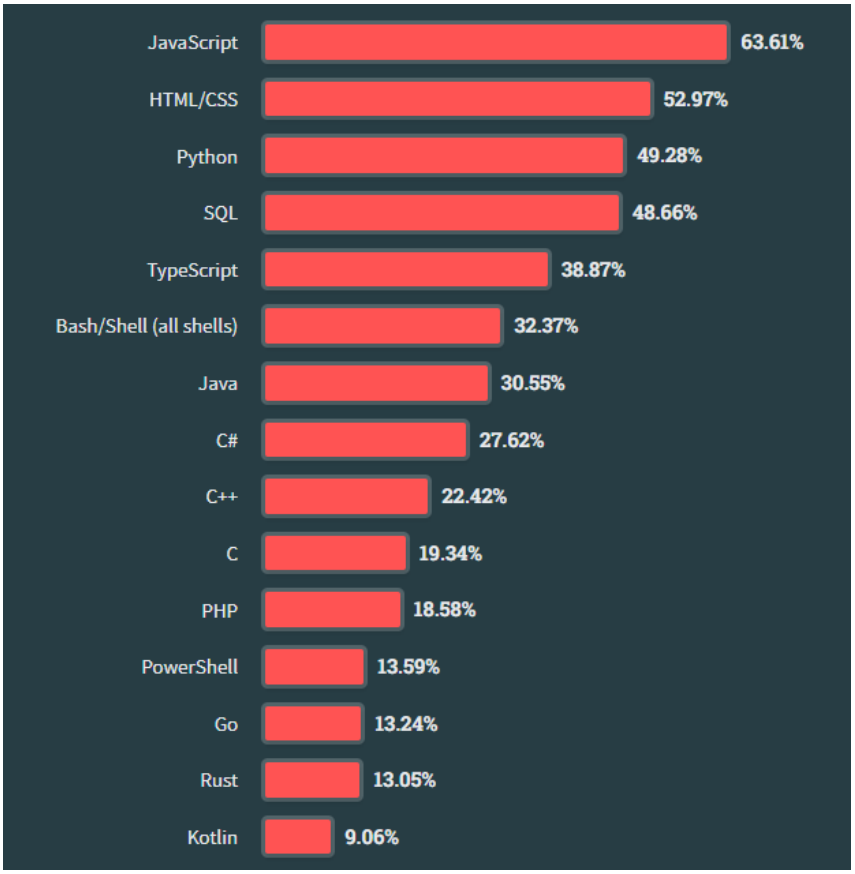
2 JavaScript

JavaScript je stěžejním tématem naší diplomové práce. Tvorba ucelené sady metodických návrhů pro učitele informatiky na 2. stupni ZŠ bude zaměřena na implementaci tohoto programovacího jazyka. V této kapitole se na něj zaměříme ze širšího pohledu a také představíme možnosti jeho využití v kontextu moderních přístupů k tvorbě webu.

V současné době se uživatelé internetu v drtivé většině případů setkávají s tzv. dynamickými webovými stránkami. Když webový server obdrží požadavek na dynamickou stránku, předá ji aplikačnímu serveru, který má za úkol dokončit načítání stránky. Aplikační server zpracuje kód stránky a dokončí načítání podle pokynů obsažených ve skriptech a značkách. Výsledkem je webová aplikace obsahující stránky dynamické nebo stránky s částečně nebo zcela nedefinovaným obsahem. Vzhled těchto stránek je následně určován aplikačním serverem na základě uživatelských pokynů během jejich používání. Dynamičnost takových stránek zajišťuje JavaScript (Adobe, 2021).

Kromě jazyků HTML a CSS je pro tvorbu webových stránek využíván také JavaScript, který přináší dynamiku a rychlou odezvu webových stránek. Poslední dobou JavaScript prochází rychlým rozvojem, přičemž se stává silným a všestranným programovacím jazykem (Pehlivanian a Nguyen, 2014). JavaScript poprvé představil Brendan Eich v roce 1995, který ho zamýšlel použít pouze jako doplněk jazyka Visual Basic od společnosti Microsoft. Původně byl navržen pro webový prohlížeč Netscape 2, kde se používal pro jednoduchou validaci formulářů a drobné manipulace s obsahem stránek. V roce 1997 se stal JavaScript standardem ECMA-262, poté co jej společnost Netscape předala organizaci ECMA. Následně organizace Mozilla převzala vývoj pro prohlížeč Firefox. Jazyk prošel postupným vývojem, přičemž byly vydány různé verze s úpravami a přidáním modernějších funkcí (W3Schools, 2024a).

Již v prvních letech své existence se JavaScript stal mimořádně populárním a předčil v popularitě všechny webové skriptovací programovací jazyky, včetně jeho největšího konkurenta VBScript od Microsoftu (Žára, 2015). Jeho výjimečné postavení v rámci programovacích jazyků získává především díky moderním webovým prohlížečům, které JavaScript implementují automaticky (Pehlivanian a Nguyen, 2014). V současné době je to jeden z nejoblíbenějších a nejpoužívanějších programovacích jazyků na světě. Podle průzkumu Stack Overflow se JavaScript v roce 2023 umístil již jedenáctý rok v řadě na první příčce jako nejpoužívanější programovací, skriptovací nebo značkovací jazyk (viz Graf 1) (Stack exchange, 2024).



Graf 1: Nejpoužívanější programovací, skriptovací a značkovací jazyky za rok 2023
(Zdroj: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies)

JavaScript se rozvíjí rychleji než jiné programovací jazyky a velké společnosti jako Netflix, LinkedIn, Uber a PayPal staví svoje aplikace na JavaScriptu. Tento jazyk je totiž vhodný jak pro vývoj front-endu, tak i pro vývoj back-endu. Dlouhou dobu se používal pouze v prohlížečích k vytváření interaktivních webových stránek. Dnes má obrovskou podporu komunity a investují do něj velké společnosti jako Meta a Google. Pomocí JavaScriptu je také možné vytvářet plnohodnotné webové nebo mobilní aplikace a také síťové aplikace v reálném čase, jako jsou chaty a služby pro streamování videa, nebo dokonce hry (Programming with Mosh, 2018).

Jak již bylo zmíněno, JavaScript byl původně navržen pouze pro běh v prohlížečích, takže každý prohlížeč má v sobě zabudovaný tzv. JavaScriptový engine, který dokáže spustit kód JavaScriptu. Například v prohlížečích Mozilla Firefox a Google Chrome jsou JavaScriptové enginy SpiderMonkey a V8. V roce 2009 softwarový inženýr Ryan Dahl dokázal zkombinovat engine V8 s programem napsaném v jazyce C++ a vytvořil tak Node.js. Díky tomu lze jednoduše spouštět kód JavaScriptu i mimo prohlížeč, což umožnilo vytvářet také back-end pro webové a mobilní aplikace (Programming with Mosh, 2018).

2.1 Syntaxe

JavaScript, stejně jako všechny ostatní programovací jazyky, dodržuje přísnou strukturu psaní kódu, známou jako syntaxe. Vzhledem k tomu, že počítače nemají lidskou schopnost intuitivně porozumět obecným myšlenkám, je nezbytné vyjádřit své požadavky přesně tak, jak to vyžaduje počítač. To znamená, že počítač bude schopen správně interpretovat programátorovi pokyny pouze v případě, že budou napsány v souladu s přesnou očekávanou formou (Moritz, 2018). Zde je příklad syntaxe v jazyce JavaScript:

```
var pozdrav = "Ahoj!";
```

Tato část kódu se označuje jako JavaScriptový příkaz a obvykle se ukončuje středníkem `;`, i když to není nutností. Použití klíčového slova `var` na začátku informuje počítač (nebo spíše JavaScriptový interpret prohlížeče), že následující slovo `pozdrav` bude proměnnou. Symbol rovnítka `=` pak interpretu sděluje, že hodnotu `Ahoj!` přiřazujeme do proměnné `pozdrav` pro pozdější použití. Přiřazení vždy obsahuje přesně jednu proměnnou na levé straně rovnítka. Například `var x = 2 + 2;` je platný JavaScriptový kód, zatímco `2 + 2 = var x;` nebo `2 + 2 = x;` jsou oba neplatné (Moritz, 2018).

JavaScript rozlišuje malá a velká písmena. To znamená, že klíčová slova, proměnné, názvy funkcí a další identifikátory musí obsahovat konzistentní kapitalizaci písmen. Například klíčové slovo `while` se musí psát `while`, nikoli `While` nebo `WHILE` (Flanagan, 2020). V názvech proměnných není dovoleno používat mezery. V ideálním případě by také měly být psány tzv. „velbloudím písmem“ (camelCase), což znamená, že začínají malým písmenem a každé další slovo v názvu proměnné začíná velkým písmenem. Termín camelCase se používá kvůli vizuální podobnosti velkých písmen uprostřed slova, které trochu vypadají jako hrb(y) na velbloudím hřbetě (Moritz, 2018).

Komentáře

JavaScript podporuje dva styly komentářů. Jakýkoli text mezi znakem `//` a koncem řádku je považován za komentář a interpret jej ignoruje. Jakýkoli text mezi znaky `/*` a `*/` je rovněž považován za komentář a mohou přesahovat na více řádků (Flanagan, 2020). Například:

```
//Tohle je jednořádkový komentář
/*Tohle je také komentář,
```

```
ale víceřádkový
*/
var a = 4; //V ukázkách kódu se bude takto značit konzolový výstup
```

Identifikátory a rezervovaná slova

Identifikátor je jednoduše řečeno název. V JavaScriptu se identifikátory používají především k pojmenování proměnných a funkcí a musí začínat písmenem, podtržítkem `_` nebo znakem dolaru `$` (Flanagan, 2020). Například:

```
moje_prvni_promenna
mojePrvniFunkce
$promennaCislo1
```

JavaScript si vyhrazuje řadu identifikátorů jako tzv. rezervovaná slova. Tato slova nelze použít jako identifikátory (názvy proměnných, funkcí apod.). Dnes je takových slov okolo šedesáti a patří mezi ně například `const`, `default`, `false`, `long`, `return` nebo `this` (W3Schools, 2024b).

Datové typy, hodnoty a proměnné

Počítačové programy pracují tak, že manipulují s hodnotami, jako je číslo `3,14` nebo text `"Ahoj světe!"`. Druhy hodnot, které lze v programovacím jazyce reprezentovat a manipulovat s nimi, se nazývají datové typy a jednou z nejzákladnějších vlastností programovacího jazyka je množina typů, které podporuje. Když program potřebuje uchovat nějakou hodnotu pro budoucí použití, přiřadí ji do proměnné. Proměnná definuje symbolický název hodnoty a umožňuje na hodnotu odkazovat jménem. Způsob, jakým proměnné fungují, je další základní vlastností každého programovacího jazyka (Flanagan, 2020).

JavaScriptové datové typy lze rozdělit do dvou kategorií: primitivní typy a objektové typy. Mezi primitivní typy patří čísla, textové řetězce a pravdivostní hodnoty (boolean). Speciální hodnoty jazyka JavaScript `null` a `undefined` jsou primitivními hodnotami, ale nejsou to čísla, řetězce ani pravdivostní hodnoty. Obě tyto hodnoty jsou považovány za jediné členy svého vlastního speciálního typu (Flanagan, 2020).

Jakákoli hodnota JavaScriptu, která není číslo, řetězec, boolean, null nebo undefined je objekt. Objekt je kolekce vlastností, kde každá vlastnost má název a hodnotu. Obyčejný objekt je neuspořádaná kolekce pojmenovaných hodnot. JavaScript také definuje speciální druh

objektu, známý jako pole, který představuje uspořádanou kolekci očíslovaných hodnot. Pole mají speciální vlastnosti, které je odlišují od běžných objektů. JavaScript definuje další speciální druh objektu, a tím je funkce. Funkce je objekt s přiřazeným kódem, který se provede až po zavolání této funkce. Stejně jako pole se i tento objekt chová jinak než ostatní druhy objektů (Flanagan, 2020).

Číslo

Čísla jsou jedním z primitivních datových typů. Mohou být kladná nebo záporná (nebo 0, která není ani kladná, ani záporná). Mohou mít desetinná místa nebo ne, ale čísla se v konzoli nikdy nezobrazují s uvozovkami. Zde je několik jednoduchých příkladů čísel: 7; 65423; 1.65; -9574; -0.25253 (Moritz, 2018).

Běžné operace s čísly:

```
10 + 2; //Znaménko plus (+) slouží ke sčítání
10 - 2; //Znaménko mínus (-) slouží k odečítání
10 * 2; //Hvězdička (*) slouží k násobení
10 / 2; //Lomítko (/) slouží k dělení
```

Rozšířené přiřazení jsou operátory spojené se znaménkem rovnosti, které slouží jako zkrácený způsob přičítání proměnné k sobě samé:

```
hodnota += 2; //Stejné jako hodnota = hodnota + 2;
hodnota -= 2; //Stejné jako hodnota = hodnota - 2;
hodnota *= 2; //Stejné jako hodnota = hodnota * 2;
hodnota /= 2; //Stejné jako hodnota = hodnota / 2;
```

Operátory inkrementace a dekrementace `++` nebo `--` jsou zkratkou pro zadání čísla a přičtení/odečtení čísla 1 k němu/od něj:

```
hodnota++; //Stejné jako hodnota = hodnota + 1;
hodnota--; //Stejné jako hodnota = hodnota – 1;
```

Operace modulo `%` se používá k získání modulu (zbytku) při dělení celých čísel:

```
10 % 2; //Zbytek 0
10 % 3; //Zbytek 1
```

Textový řetězec

Snad nejběžnějším datovým typem jsou textové řetězce. Při vytváření nového řetězce se používají jednoduché `''` nebo dvojité `""` uvozovky (Moritz, 2018). Zde je několik příkladů textového řetězce:

```
"Ahoj";
'světe!';
"123456789";
```

Při vytváření nových textových řetězců je třeba věnovat velkou pozornost jednoduchým a dvojitým uvozovkám. Pokud řetězec začíná dvojitou uvozovkou, musí jí také končit. Ale samozřejmě lze v jednom řetězci zapsat oba typy uvozovek:

```
"I don't have any money.";
```

Textové řetězce se skládají z jednotlivých částí, které se nazývají znaky. Nelze říci, že jsou tvořeny písmeny, protože jak již bylo zmíněno, řetězce mohou obsahovat také čísla, a dokonce i symboly (Moritz, 2018).

Někdy nastane situace, kdy je potřeba napsat obě uvozovky v jednom textovém řetězci:

```
"He said "I'm your father!"."; //SyntaxError
```

Takový příkaz by vedl syntaxové chybě. Zpětné lomítko `\` slouží k tzv. escapování (textový řetězec může být obklopen jednoduchými nebo dvojitými uvozovkami). Obojí je v pořádku. Pokud je potřeba použít stejný druh uvozovek, který se nachází na vnějších stranách i uprostřed, musí se escapovat pomocí zpětného lomítka (Moritz, 2018). Například:

```
"He said \"I'm your father!\".";
```

Při práci s textovými řetězci má znaménko plus `+` zvláštní význam. Místo pro sčítání se používá pro spojování:

```
"Hello" + " " + "World" + "!"; //Hello World!
```

Tento příkaz by vrátil řetězec, což je datový typ. Pokud se sečtou dvě nebo více čísel, interpret vrátí číselnou hodnotu. Pokud se sečtou dva nebo více řetězců, interpret vrátí řetězec. Pokud se sečte řetězec s číslem, interpret automaticky převede číslo na řetězec.

To však nefunguje s ostatními operátory, protože nelze odčítat, násobit ani dělit znaky (Moritz, 2018). Například:

```
"5" + 5; //"55"
"5" - 5; //0
```

Pravdivostní hodnota (boolean)

Třetí primitivní datový typ se nazývá pravdivostní, nebo také logická hodnota. Pravdivostní hodnoty představují pravdu nebo lež, zapnuto nebo vypnut, ano nebo ne, nic mezi tím. Existují pouze dvě možné hodnoty tohoto typu, kterými jsou rezervovaná slova `true` a `false`. K pravdivostním hodnotám se úzce vztahují operátory porovnávání, které se využívají, pokud je třeba porovnat hodnoty. Operátorů porovnávání je celkem 8 a nezáleží na tom, který z operátorů se použije a jak se použije, ale vždy interpret vrátí pravdivostní hodnotu `true` nebo `false` (Moritz, 2018).

Trojí rovnost `===` – tento typ porovnání kontroluje, zda je hodnota stejná a zda je stejný i datový typ. Pokud obojí souhlasí, typ i hodnota, interpret vrátí hodnotu `true`, v opačném případě vrátí `false`. Například:

```
7 === 7; // true
7 === "7" // false
```

Dvojitá rovnost `==` – Tento typ porovnávání se stará pouze o to, zda je stejná hodnota. Nemusí se zde tedy rovnat datové typy, aby interpret vrátil hodnotu `true`. Například:

```
7 == 7 // true
7 == "7" // true
7 == "sedm" // false
```

Nerovná se rovná se `!==` – v programování vykřičník `!` vždy znamená “ne”. Je to způsob, jak něco negovat (přivést do záporu). Takže `!==` interpret vždy vrátí opak `===`. Například:

```
7 !== 8 // true
7 !== "7" // true
7 !== 7 // false
```

Nerovná se `!=` – Tento typ porovnání vždy vrátí opak `==`. Například:

```
7 != 7 // false
7 != "7" // false
7 != "sedm" // true
```

Větší než `>` a menší než `<` – Tyto typy porovnání jsou poměrně jednoduché. Interpret vrátí hodnotu `true`, pokud je první hodnota větší `>` nebo menší `<` než hodnota druhá. Například:

```
7 > 10 // false
7 > 7 // false
7 < 10 // true
```

Větší než nebo rovno `>=` a menší než nebo rovno `<=` – Tyto typy porovnání jsou v podstatě stejné jako předchozí, ale vrátí hodnotu `true`, i když bude druhá hodnota stejná jako ta první. Například:

```
7 >= 10 // false
7 >= 7 // true
7 <= 10 // true
```

Pravdivostní hodnoty se běžně používají v řídících strukturách, jako například příkaz `if…else` provede jednu akci, pokud se pravdivostní hodnota rovná `true`, a jinou akci, pokud se rovná `false` (Flanagan, 2020). Například:

```
if (a == 4) { b = b + 1 }
else { a = a + 1 }
```

Operátor `&&` provádí logickou operaci AND. Vyhodnotí se jako pravdivá hodnota tehdy a jen tehdy, pokud jsou oba operandy pravdivé. V opačném případě se vyhodnotí jako nepravdivá hodnota. Operátor `||` je logická operace OR. Vyhodnotí se jako `true`, pokud je alespoň jeden z operandů pravdivý. A opět se setkáváme s operátorem `!`, který provádí logickou operaci NOT. Vyhodnotí se jako pravdivý, pokud má jeho operand hodnotu `false`, a vyhodnotí se jako nepravdivý, pokud má jeho operand hodnotu `true` (Flanagan, 2020). Například:

```
if ((x == 0 && y == 0) || !(z == 0)){
```

```
        // proměnné x a y jsou obě nula nebo proměnná z není nula
}
```

Null a undefined

Čtvrtý primitivní datový typ je jednoduchý: `null`. Null v podstatě znamená nic. Není to ani 0, ani `""`, ani `false`. Například:

```
var mojePromenna = null;
mojePromenna; //null
```

Je to užitečný způsob, jak nastavit hodnotu něčeho, co má být někdy v budoucnu změněno. Například při vytváření formuláře. Formulář obsahuje tři pole: jméno, věk, pohlaví. Při vytváření formuláře bude potřeba nastavit některé počáteční hodnoty pro každou z těchto proměnných. Mohly by se zde použít hodnoty `null`, a poté, co uživatel formulář vyplní, budou k dispozici hodnoty, které lze použít. Pokud by uživatel nevyplnil všechny hodnoty, není třeba se o ně starat, protože jsou již nastavené na hodnotu `null` (Moritz, 2018).

Undefined – Pátý primitivní datový typ se nazývá `undefined` (nedefinován). Jako datový typ `undefined` znamená, že proměnné nebyla přiřazena žádná jiná hodnota, dokonce ani `null`. Jedná se o výchozí hodnotu proměnné (Moritz, 2018). Například:

```
var novaPromenna;
novaPromenna; //undefined
```

Kondicionály

Kondicionály neboli podmíněné příkazy v programování slouží k provedení určitých bloků kódu na základě dané podmínky. Výsledkem podmínky (například operátoru porovnání jako `x > y`) je pravdivostní hodnota pravda nebo nepravda (`true` nebo `false`). Pokud je pravdivostní hodnota pravda, blok kódu se provede. V opačném případě interpret blok kódu přeskočí (neprovede se) (Moritz, 2018).

Nejčastějším příkladem podmíněného příkazu v jazyce JavaScript je příkaz `if`. Tento příkaz kontroluje pravdivostní hodnotu uvnitř závorky a určuje, zda se má kód uvnitř bloku spustit, nebo ne. Hodnota v podmínce však nemusí nutně obsahovat pouze `true` nebo `false`. Například jakékoliv číslo kromě nuly se rovná pravdě, nula se potom vyhodnocuje jako

nepravda (Moritz, 2018). V dalším příkladu je také napsána statická metoda `console.log()`, pomocí které se vypisuje text do konzole (Mozilla.org, 2024). Například:

```
if(5) { console.log("Kód se provede."); }
if(0) { console.log("Kód se neprovede."); }
```

Příkaz `if` se hodí tam, kde je potřeba, aby se nějaká část kódu spustila pouze v případě, že je daná podmínka pravdivá. Je však dobré mít i záložní plán pro případ, že podmínka pravdivá nebude. Tehdy se dostáváme k použití podmínky `else`. Pokud je podmínka pravdivá, interpret spustí blok kódu uvnitř `if` části. Pokud se podmínka pravdě rovnat nebude, interpret spustí kód uvnitř části `else` (Moritz, 2018). Například:

```
if(0) { console.log("Kód se neprovede."); }
else { console.log("Kód se provede."); }
```

Příkaz `if…else` vyhodnotí příkaz a v závislosti na výsledku provede jednu ze dvou částí kódu. Pokud je ale potřeba rozhodovat z více než jen dvou možností, používá se příkaz `else if` (Flanagan, 2020). Například:

```
if (a == 1) {
        //Proveď tento blok kódu
} else if (a == 2) {
        //Proveď tento blok kódu
} else {
        //Proveď tento blok kódu
}
```

Pole

Pole v jazyce JavaScript (a dalších programovacích jazycích) je uspořádaná kolekce hodnot. Každá hodnota se nazývá prvek a každý prvek má v poli číselnou pozici, která se nazývá index. Prvek pole může být libovolného datového typu a různé prvky téhož pole mohou být také různých typů. Tyto prvky mohou být dokonce objekty nebo jiná pole, což umožňuje vytvářet komplexní datové struktury. Jednoduše řečeno se pole používá k uložení více hodnot do jedné proměnné. Nejjednodušší způsob vytvoření pole je přiřazení čárkou

oddělených prvků v hranatých závorkách `[]` do proměnné (Flanagan, 2020). Příklad pole může vypadat následovně:

```
var barvy = ["zelená", "modrá", "červená"];
```

Název proměnné je obvykle v množném čísle, a má určitou spojitost s hodnotami v ní, protože se jedná o kolekci těchto položek. Každá z těchto položek v poli má přiřazené číslo. Toto číslo představuje tzv. index dané položky. Index pole je číslo představující pozici libovolné položky v seznamu. Pozice se začínají počítat od čísla 0 (nikoli od čísla 1), takže zdánlivě první položka v seznamu má indexovou pozici 0. Pole má také délku, která představuje počet jeho položek. Jakmile je pole vytvořené, lze ke každé jeho položce přistupovat zvlášť pomocí závorek s indexem konkrétní položky. Tato indexová pozice položky se zapisuje do hranatých závorek (Moritz, 2018). Například:

```
barvy[0]; //"zelená"
```

V jazyce JavaScript jsou pole dynamická, což znamená, že se zvětšují nebo zmenšují podle potřeby a není třeba deklarovat pevnou velikost pole při jeho vytváření ani při změně jeho velikosti. Pro manipulaci s obsahem polí se používají tzv. metody (Flanagan, 2020). Například:

```
barvy.push("bílá");
barvy.pop();
```

Cykly

Cyklus je v jazyce JavaScript blok kódu, který se opakuje stále dokola, dokud platí určitá podmínka. Prvním takovým příkladem je cyklus `while`. Při provádění příkazu `while` interpret nejprve vyhodnotí výraz. Pokud je hodnota výrazu nepravdivá, interpret tělo cyklu neprovede a přeskočí na další příkaz v programu. Pokud je naopak výraz pravdivý, interpret tělo cyklu provede a bude jej opakovat tak dlouho, dokud bude výraz pravdivý. Výrazem, nebo také podmínkou, je cokoli, co má program před každým opakováním zkontrolovat. Kdykoli je tato podmínka pravdivá, program ví, že má cyklus opakovat. Jakmile podmínka přestane být pravdivá, program přestane blok kódu opakovat (Moritz, 2018). Například:

```
var i = 0;
while(i < 3) { console.log(i); i++; } //0, 1, 2
```

Protože při použití cyklu `while` lze snadno udělat chybu, a mohlo by se stát, že by se program opakoval donekonečna, mnohem častěji se v programech setkáváme s cyklem `for`, který je cyklu `while` dosti podobný, ale má zabudovaný proměnný čítač. Kdykoli je předem jasné, kolik iterací smyčky bude potřeba provést, je vhodnější místo `while` použít cyklus `for`. Syntaxe tohoto cyklu je v podstatě stejná jako u `while`, ale je zkrácena na tři části, které jsou odděleny středníky (Moritz, 2018). Příklad tohoto cyklu může vypadat například takto:

```
for (var i = 0; i < 3; i++){ console.log(i); } //0, 1, 2
```

Funkce

V programování je funkce oddělený blok kódu, který je definován jednou a lze zavolat k provedení určitého úkolu kolikrát bude třeba. Funkce jsou v jazyce JavaScript parametrizované, což znamená, že definice funkce může obsahovat seznam identifikátorů, známých jako parametry, které fungují jako lokální proměnné pro tělo funkce (Flanagan, 2020). Funkci je třeba nejprve deklarovat:

```
function mojePrvniFunkce(parametr) { console.log(parametr) }
```

Když je funkce deklarována (zavedena do kódu), je připravena na pozdější použití. `function` je klíčové slovo v jazyce JavaScript a říká interpretu, že to, co bude následovat, bude tzv. vlastní funkce (taková, kterou jsme vytvořili sami, nikoliv vestavěná a předem vytvořená). Název stylem camelCase `mojePrvniFunkce` je vlastní název funkce. Závorky `()` jsou nutné vždy, kdy se funkce deklaruje, protože je někdy třeba do funkce přidat jeden nebo více parametrů, které se později předají při volání funkce jako argument. Složené závorky `{}` obklopují tělo funkce neboli blok kódu, který se provede při jejím volání (Moritz, 2018):

```
 myFirstFunction("Ahoj světe!");
```

Vestavěné funkce – některé funkce jsou v JavaScriptu již zabudovány a dají se kdykoliv použít. Mezi vestavěné funkce patří například tyto:

```
alert("Ahoj světe!");
prompt("Zadej svoje jméno:");
confirm("Jsi si jistý?");
console.log("Ahoj světe!");
```

```
Math.random();
Math.floor(9.75);
```

Vestavěné funkce `alert()`, `prompt()` a `confirm()` jsou tzv. vyskakovací okna, nebo také hlášky. Funkce `alert()` je ve své podstatě jenom upozornění pro uživatele o provedení nějaké akce, tudíž s ní uživatel nemůže interagovat. Uživatel může hlášku pouze potvrdit tlačítkem „OK“, tím úloha funkce `alert()` končí. Vestavěná funkce `prompt()` je také brána jako hláška, s tím rozdílem, že s ní může uživatel pracovat, konkrétně do ní může napsat text. Text zapsaný uživatelem se většinou ukládá do hodnoty proměnné a v programu se s ní dále pracuje. Funkce `confirm()` je poslední z vyskakovacích okének, která vyžaduje dvoustavovou odpověď od uživatele. Uživatel si může vybrat ze dvou tlačítek, kde první tlačítko „OK“ vrací hodnotu `true`, a druhé tlačítko „Zrušit“ vrací hodnotu `false`. Funkce `confirm()` se většinou využívá při rozhodování, zda chce uživatel pokračovat v určitém úkonu, jako třeba odhlášení se ze stránky bez uložení, prodloužení relace na stránce apod. (Kantor, 2024).

Metody a vlastnosti

Vlastnost je pojmenovaná hodnota, která je připojena k objektu. Objekty jsou důležitou součástí JavaScriptu. Objekt v jazyce JavaScript je kolekce vlastností, jako například:

```
Math.random();
```

`Math` je objekt a tečka `.` interpretu říká, že další slovo `random` je vlastnost. Protože za slovem `random` následují závorky, znamená to, že `random` musí být funkce. Když je vlastnost funkcí, existuje pro ni speciální název: metoda. Dalším druhem objektu je textový řetězec a ten má vlastnosti jako `length`, které o něm zjistí určité informace. Textové řetězce obvykle nemohou mít metody nebo vlastnosti, protože nejsou objekty. V JavaScriptu jsou však metody a vlastnosti dostupné i textovým řetězcům, protože JavaScript při provádění metod a vlastností považuje řetězce za objekty (W3Schools, 2024c). Objekty mají také metody, které umožňují s nimi manipulovat. Při práci s textovými řetězci jsou příkladem metody `.concat()` nebo `.repeat()` (Moritz, 2018):

```
"Ahoj".concat(" světe."); //"Ahoj světe."
```

Dalšími metodami jsou například `.toUpperCase()` (převede všechna písmena v textovém řetězci na velká) nebo `.toLowerCase()` (převede text na malá písmena).

Objektový model dokumentu (DOM)

Objektový model dokumentu, zkráceně DOM, se dá popsat jako určité rozhraní, pomocí kterého se lze odkazovat na jednotlivé HTML elementy na webové stránce (Štráfelda, 2024). Na jednotlivé HTML elementy se dá odkazovat pomocí metod `getElementById()`, `getElementsByClassName()`, `getElementsByTagName()` apod. Každá z těchto metod se odkazuje na jiné atributy jednotlivých elementů, například metoda `getElementById()` se odkazuje na jedinečný identifikátor, zkráceně id, který je přiřazený určitému elementu. Další zmíněné metody vybírají hned několik elementů, a to buď podle třídy v případě `getElementsByClassName()` nebo podle tagů v případě `getElementsByTagName()`. Obě tyto metody vrací hodnoty v poli, tudíž je nutné se na jednotlivé prvky v poli odkazovat stejně jako u normálních polí pomocí hranatých závorek `[]`, do kterých se vepisuje jejich pozice v poli (GeeksforGeeks, 2024).

2.2 Využití JavaScriptu při tvorbě webových stránek a aplikací

Web existuje déle než 25 let a prošel rozmanitými fázemi, začínaje skvělým počátkem, následným ekonomickým propadem, až po znovuzrození díky inovacím a neustálému vývoji. Je jisté, že web zůstane nedílnou součástí našich životů i v dalších letech. Navíc si našel cestu do různých druhů zařízení, včetně chytrých telefonů, tabletů a dalších. Pohled na konstrukci webu odhaluje, kolik rozhodnutí a odborných znalostí bylo zapotřebí během jeho výstavby. Webové stránky představují více než pouhý soubor kódů a obrázků. Často začínají obchodním plánem nebo jiným jasně stanoveným posláním. Před spuštěním je nezbytné vytvořit a uspořádat obsahovou koncepci, provést průzkum, navrhnout design od obecných cílů až po nejmenší detaily, napsat kód a propojit vše s tím, co probíhá na serveru, aby spolu vše bezchybně fungovalo (Robbins, 2018).

Při využívání internetu se propojují dva hlavní druhy počítačů: servery, které uchovávají a poskytují informace, a klienti, kteří tato data vyhledávají a prezentují je uživatelům. Internet je v zásadě koncipován jako prostředek, který není závislý na klientském hardwaru. Internetové dokumenty tedy obsahují pouze text, zatímco veškeré další obsahy, jako jsou obrázky, videa a jiné grafické prvky jsou uloženy v oddělených souborech a je na ně odkazováno pomocí

odkazů. Internetové prohlížeče, jako je Microsoft Edge, Google Chrome, Opera nebo Firefox, jsou poté používány k zobrazení internetových dokumentů na straně klienta (Laurenčík, 2019).

Větší a známé webové stránky jsou vytvářeny týmy složenými z desítek, stovek nebo dokonce i tisíců pracovníků. Existují však i menší weby, které úspěšně vytvářejí a spravují týmy s malým počtem členů. Vytvořit kvalitní web může také jednotlivec. Právě v tom spočívá krása tvorby webových stránek. Významnou část procesu vytváření webových stránek tvoří práce a řešení problémů spojených s dokumenty, styly, skripty a obrázky, které utvářejí strukturu webových stránek. Celý proces vývoje lze rozdělit do dvou hlavních kategorií: vývoj frontendu a vývoj backendu. Tyto úkoly mohou být svěřeny specializovaným odborníkům, ale stejně tak je běžné, že jedna osoba nebo tým zvládne obě části vývoje. Takové osoby se potom nazývají fullstack vývojáři (Robbins, 2018).

2.2.1 Frontend

Vývoj frontendu se týká všech aspektů návrhu, které jsou viditelné v prohlížeči nebo přímo s ním souvisejí. Sem patří HTML, CSS a JavaScript, které jsou nezbytnými znalostmi pro práci jako webový vývojář. Za vývojem frontendu stojí tyto technologie:

Autorizace/značkování (HTML)

Autorizace představuje proces přípravy obsahu pro prezentaci na webu, konkrétně značkování obsahu pomocí HTML značek, které popisují jeho obsah a funkce. Jazyk HTML (HyperText Markup Language) je značkovací jazyk využívaný k vytváření dokumentů webových stránek (Robbins, 2018). Jeho počátky sahají až do roku 1980, kdy fyzik Tim Berners-Lee představil systém ENQUIRE pro vzájemné sdílení dokumentů vědců CERNu. V roce 1990 specifikoval jazyk HTML a vyvinul pro něj prohlížeč spolu se serverovým softwarem (Denis, 2024). Od té doby prošel jazyk HTML postupným zdokonalováním až po aktuální verzi HTML5. HTML není programovací jazyk, nýbrž značkovací jazyk, což znamená, že slouží k identifikaci a popisu různých částí dokumentu, jako jsou nadpisy, odstavce a seznamy. Značky definují základní strukturu dokumentu, což si lze představit jako podrobný a strojově čitelný plán. V souborech typu HTML jsou uloženy řídící příkazy zajišťující správné zobrazení internetového dokumentu v prohlížeči. Pro psaní kódu v HTML nejsou potřeba programátorské dovednosti, pouze trpělivost a logické uvažování (Laurenčík, 2019).

Stylování (CSS)

Každý HTML soubor by měl také obsahovat informace o formátování. Formátování lze provést několika způsoby: údaje o velikosti písma, barvě a zarovnání lze zapsat přímo do jednotlivých HTML značek, nebo lze informace o formátování zapsat do samostatného souboru pomocí kaskádových stylů (CSS) a v HTML souboru se na ně pouze odkazovat. Zatímco jazyk HTML se využívá k popisu obsahu webové stránky, kaskádové styly určují, jak by tento obsah měl vypadat. Prezentace stránky, tedy způsob, jakým se stránka jeví koncovému uživateli, je ovládána pomocí jazyku CSS. Tento jazyk řídí písma, barvy, obrázky na pozadí, zarovnání, rozložení stránky a další. S pomocí kaskádových stylů lze na stránku přidat i speciální efekty a základní animace. Specifikace jazyka CSS rovněž poskytuje prostředky pro ovládání prezentace dokumentů v různých situacích mimo prostředí prohlížeče, jako je například při tisku nebo při hlasitém předčítání pomocí čtečky obrazovky. I když je možné publikovat webové stránky pouze pomocí jazyka HTML, je velice pravděpodobné, že bude potřeba implementovat styly, aby vzhled stránky nebyl závislí na výchozích stylech prohlížeče (Robbins, 2018).

JavaScript a skriptování DOM

JavaScript představuje skriptovací jazyk, který přináší interaktivitu a dynamiku do webových stránek. Některé z jeho možností zahrnují:

- Kontrola správnosti vstupů ve formulářích;
- Změna stylů pro určitý prvek nebo celý web;
- Automatické načítání dalšího obsahu při rolování;
- Ukládání informací o uživatelských preferencích v prohlížeči;
- Vytváření uživatelských rozhraní, například vložené přehrávače videa nebo speciální formulářové prvky.

V kontextu jazyka JavaScript se často používá termín DOM skriptování. DOM je zkratkou pro Document Object Model a označuje standardizovaný seznam prvků webových stránek, ke kterým lze přistupovat a s nimiž lze manipulovat pomocí jazyka JavaScript (nebo jiného skriptovacího jazyka). Frontendoví vývojáři mohou potřebovat znalost některého z JavaScriptových frameworků, jako jsou například React Native, Angular nebo Vue, které automatizují část vývojového procesu. Také může být nezbytná zručnost v AJAXu (Asynchronous JavaScript And XML), což je technika umožňující načítání obsahu na pozadí

a umožňující plynulou aktualizaci stránky bez jejího opětovného načítání. Skriptování webových stránek vyžaduje určité tradiční znalosti v oblasti počítačového programování. Mnoho vývojářů webových aplikací má vysokoškolské vzdělání v oboru informatiky, ale běžně se setkáváme i s případy, kdy jsou vývojáři samouky. Pro lidi, kteří chtějí pracovat jako weboví vývojáři, je znalost JavaScriptu nezbytná (Robbins, 2018).

2.2.2 Backend

Vývojáři backendu se zaměřují na server, včetně aplikací a databází, které na něm běží. Jsou zodpovědní za instalaci a konfiguraci serverového softwaru. Jejich úkolem je znalost alespoň jednoho, pravděpodobně však více programovacích jazyků na straně serveru, jako jsou PHP, Ruby, .NET, Python nebo JSP, aby mohli vytvářet aplikace s funkcemi požadovanými pro webové prostředí. Tyto aplikace se zabývají úkoly a funkcemi, jako je zpracování formulářů, správa obsahu a online nakupování apod. Vývojáři backendu navíc musejí mít znalosti o konfiguraci a údržbě databází, které uchovávají všechna data webu, například obsah, který se vkládá do šablon, uživatelské účty, seznamy produktů a další. Mezi obvyklé databázové jazyky patří MySQL, Oracle nebo SQL Server. Existuje mnoho dalších rolí, které se podílejí na vytváření a údržbě webových stránek. Několik běžných rolí, které spadají mimo oblast označovanou jako „webdesign“, zahrnuje produktového manažera, projektového manažera, specialistu na SEO taktiky a výrobce multimédií (Robbins, 2018).

2.3 Vývojová prostředí

Vždy lze nalézt prostředky, které by mohly usnadnit dosažení osobních cílů. Například spisovatelé využívají textové editory, jako je Microsoft Word nebo Google Docs. Účetní se opírají o tabulkové procesory jako jsou Microsoft Excel a Google Sheets. Programátoři zase využívají integrovaná vývojová prostředí (IDE). Integrované vývojové prostředí slouží k sjednocení různých prvků procesu psaní počítačového programu. Tím, že kombinují úpravu zdrojového kódu, kompilaci spustitelného souboru a ladění do jediné aplikace, integrovaná vývojová prostředí programátorům znatelně ulehčují práci (Codecademy, 2024). V dnešní době existuje rozsáhlá škála různých druhů integrovaných vývojových prostředí. Některá fungují výhradně online, zatímco jiná mohou být používána buď v lokálním prostředí nebo taktéž online s mírnými úpravami. Před zavedením vývojových prostředí programátoři často využívali textové editory, jako například poznámkový blok, k zápisu svého kódu. Následně ukládali tento kód s použitím specifických přípon pro dané programovací jazyky,

jako je například .php pro soubory v jazyce PHP nebo .js pro JavaScriptový kód. Dnes existuje spousta vývojových prostředí, jako jsou například Visual Studio Code, Atom, Replit, PyCharm, NetBeans nebo Eclipse (Olawanle, 2022).

Vývojová prostředí obvykle zahrnují editor kódu, překladač nebo interpret a ladící program (debugger), které jsou dostupné prostřednictvím jediného grafického uživatelského rozhraní (GUI). Uživatel pracuje s editorem kódu, kde píše a upravuje zdrojový kód. Překladač převádí zdrojový kód do čitelného jazyka, který je spustitelný na počítači. Ladící program testuje software a řeší případné problémy nebo chyby. Vývojové prostředí může také zahrnovat funkce, například programovatelné editory, objektové a datové modelování, testování jednotek, knihovny zdrojového kódu a nástroje pro automatizaci psaní kódu. Uživatelé mohou prostřednictvím rozhraní vývojového prostředí postupně kompilovat a spouštět kód a sjednoceně spravovat změny ve zdrojovém kódu. IDE jsou často navržena tak, aby se integrovala s knihovnami třetích stran, mezi které patří například GitHub (Gillis, 2018).

3 Metodika a didaktika

Před praktickou částí je třeba zmínit některé didaktické zásady, výukové metody a organizační formy, které by se daly využít v hodinách zaměřených na výuku algoritmizace a programování. Vybrané výukové metody, didaktické zásady a organizační formy následně využijeme v rámci tvorby našich metodických návrhů.

3.1 Výukové metody, didaktické zásady a organizační formy

Ve škole jako výchovně-vzdělávací instituci se změny vyskytují jen výjimečně, především tedy při zásadních reformách. V případě využívaných metod při výuce je tomu však jinak, jelikož jsou tyto metody úzce svázány s učiteli a žáky, a tak se jednodušeji přizpůsobují aktuálním potřebám. Výuková metoda představuje ve výuce *„určitý dynamický prvek, který se ve srovnání s obsahem a organizačními formami relativně rychleji mění a přizpůsobuje novým cílům a okolnostem.“* (Maňák a Švec, 2003, s. 9). Výukovou metodu si můžeme představit jako didaktický prostředek s určitým postupem, pomocí kterého bychom měli dosáhnout výchovně-vzdělávacího cíle. Nejedná se ovšem pouze o činnosti učitele, nebo pouze o činnosti žáka, jelikož se obě tyto činnosti ve vyučovacím procesu navzájem propojují a doplňují. Výukové metody jsou však jen jedním z několika prvků vzdělávacího systému a jsou integrovány do celkové koncepce výuky, kde mohou být maximálně efektivní (Maňák a Švec, 2003).

Také je třeba od sebe odlišit pojmy vyučování a učení, které jsou základními prvky pedagogické interakce. Tyto procesy se vzájemně propojují v sociálním kontextu třídy. Vyučování je činnost učitele, při které stimuluje odpovídající učební aktivity žáků, založené na určitých výukových cílech. Učitel tak může s žáky diskutovat o učivu, zadávat jim samostatnou práci, klást důraz na klíčové informace apod. Pomocí těchto postupů učitel navozuje učení žáků, kteří při tomto procesu získávají znalosti, dovednosti a návyky a rozvíjejí tak své schopnosti. Při tak komplexním procesu, jako je učení, se na jeho průběhu podílejí různé individuální faktory, jako například pozornost, představivost, paměť nebo emoční vyspělost (Maňák a Švec, 2003).

Pomocí dodržování didaktických zásad, což jsou všeobecné doporučení pro učitele, může učitel dosáhnout maximální účinnosti při výuce. Tyto didaktické zásady se vážou na všechny aspekty výuky, včetně výukových metod, práce žáka, či didaktických materiálů. Didaktické zásady se vyvinuly na základě praktických zkušeností pedagogů, kteří ve své praxi

zaznamenávali úspěšné postupy, které vedly k pozitivním výsledkům ve výchovně-vzdělávacím procesu. Ideologie didaktických zásad sahá až k Aristotelovi, který kladl důraz na aktivitu žáka. Systematické uplatňování didaktických zásad nastalo až okolo 16. století, jako reakce na přehlížení potřeb a zájmů žáků ve středověké výuce (Zormanová, 2012a).

3.1.1 Didaktické zásady

Zásada uvědomělosti a aktivity

Tato zásada vyžaduje, aby se žáci aktivně zapojovali do vyučovacího procesu. Takový žák by měl mít zodpovědný vztah k učení, měl by sám chtít načerpat co možná nejvíce vědomostí, a tudíž pro to dělat vše, co je v jeho schopnostech. Zásada uvědomělosti se nevztahuje pouze k žákovi, ale je také důležité si uvědomit, že pro motivaci žáka k uvědomělosti a aktivitě hraje významnou roli také učitel. Učitel, který dodržuje tuto zásadu by měl mimo jiné jasně formulovat cíle vyučovacích jednotek, motivovat žáky a učit je aplikovat nově nabyté poznatky i mimo školní třídu (Zormanová, 2012a).

Tato zásada lze využít např. při programování, kdy si žáci mohou pokládat otázky jako „Proč jsem zvolil tuto metodu k vyřešení problému?“. Žáci se také mohou zamýšlet nad svými emocemi, postoji a myšlenkovými procesy při řešení samostatných cvičení. Když žáci vytvářejí svůj kód, měli by si všímat svých případných chyb a aktivně hledat způsob, jak tyto chyby napravit. Při dodržování této zásady by měl učitel dát žákům dostatek času na testování a ladění jejich kódu, aby si během tohoto procesu mohli uvědomit své rozhodnutí a úvahy, které vedly k vytvoření programu a zda by mohli program ještě zdokonalit.

Zásada spojení teorie s praxí

Při dodržování této zásady je podstatné, aby žáci nově nabyté poznatky mohli aplikovat také v reálném životě, a ne pouze ve školní třídě. Je tedy třeba ve výchovně-vzdělávacím procesu zastupovat jak teoretickou, tak i praktickou rovinu (Zormanová, 2012a). Učitel by neměl předávat poznatky pouze v kontextu školy, izolovat je od světa, ale představovat je na reálných a praktických příkladech, aby žáky přesvědčil o jejich smysluplnosti. Tento argument podporuje především fakt, že žáci se neučí novým věcem pouze ve škole, ale také mimo ni. Proto je podstatné především pro učitele informatiky, aby naučili žáky jak a kde vyhledávat relevantní informace, aby si mohli sami ověřovat informace, které se k nim dostanou (Kolářová, 2019).

Tato zásada lze při výuce algoritmizace a programování využít například na předkládání žákům praktických příkladů a projektů, které mohou využít také v reálném životě. Nemusí se nutně jednat o kód v programovacím jazyce, ale lze žákům například předložit princip fungování algoritmů, a jak jej nevědomky využívají v každodenním životě, např. přechod přes cestu (stojím, dokud nesvítí zelená) nebo vaření čaje (nejprve vodu uvařím, …). Tato zásada se dá také využít při projektově orientované výuce, kdy mají žáci za úkol vytvořit program na určité téma, určitý problém, který lze později smysluplně využít v praxi.

Zásada přiměřenosti

Tato didaktická zásada vyžaduje, aby výukové cíle, obsah učiva, použité organizační formy a výukové metody adekvátně odpovídaly vědomostem, schopnostem a dovednostem vzdělávaných žáků. Je potřeba brát v úvahu také psychické zvláštnosti jednotlivých věkových skupin (Zormanová, 2012a).

Tato zásada je při výuce algoritmizace a programování velice důležitá, jelikož pomáhá udržovat rovnováhu mezi obtížností úloh a schopnostmi studentů. Je tedy vhodné začít nejjednoduššími úkoly a postupně zvyšovat jejich obtížnost, aby se studenti mohli postupně seznamovat s novými koncepty a dovednostmi. Žákům, kteří se v programování moc neorientují, lze připravit různé podpůrné materiály, které jim práci usnadní. Naopak lze připravit složitější úlohy pro žáky, kteří by se při zpracovávání samostatného cvičení nudili. Při hodnocení a známkování jednotlivých programů žáků je také třeba přistupovat flexibilně, a i slabší žáky hodnotit lépe, a motivovat je tak ke zdokonalování se.

Zásada trvalosti

Tato zásada si klade za cíl žákovo trvalé osvojení znalostí a dovedností. Pokud je učivo nedostatečně osvojeno, je možné, že jej žák brzy zapomene. Tudíž je potřeba učivo provázat s již osvojenými vědomostmi a nestačí se jej pouze učit nazpaměť. Žák by si měl dokázat učivo kdykoliv vybavit, čemuž může učitel pomoct již zmíněným provazováním nového učiva s již osvojeným a jeho pravidelným opakováním (Zormanová, 2012a).

Tato zásada je pro výuku programování velice důležitá, jelikož je třeba s žáky pravidelně cvičit a opakovat jednotlivé příkazy a koncepty v programování. Dostatečné opakování žákům pomůže získat pevný základ a v budoucnu posílit své dovednosti. Tato zásada se úzce proplétá se zásadou přiměřenosti, jelikož je pro žákovo upevnění učiva třeba začít od základních příkazů a pokračovat k pokročilejším, při čemž by se základní příkazy měli

co nejvíce opakovat, aby je žáci dokázali automaticky využívat. Práce ve škole lze také promítat do života mimo školu, proto je třeba žáky motivovat, aby si zkoušeli programovat i doma, doporučit jim různé webové stránky, kde mohou najít návody, rady, inspiraci apod. Když budou žáci vystaveni programování i ve volném čase, mají mnohem větší šanci si tuto schopnost osvojit a dlouhodobě udržet.

Zásada soustavnosti

Tato zásada se zakládá na osvojování dovedností a znalostí systematicky. Pro žáky je důležité, aby na sebe probíraná látka logicky navazovala. Je ale nutné probíranou látku stupňovat od jednoduchého ke složitějšímu, aby se žákovi schopnosti a dovednosti mohli dále rozvíjet (Zormanová, 2012a).

Tato zásada lze využít například při vysvětlování nových příkazů ve vztahu na ty předchozí. Je tedy opět třeba nejprve žáky seznámit se základními příkazy, na které se budou dále nabalovat složitější a komplexnější. Je vhodné žákům zadávat samostatné cvičení, aby si mohli sami vyzkoušet provázanost příkazů, zamyslet se nad nimi a monitorovat svůj pokrok a identifikovat oblasti, na kterých musejí ještě pracovat. Pro tuto zásadu je tedy podstatné dodržování systematického rozvoje dovedností, kterého lze dosáhnout pravidelným procvičováním a zpětnou vazbou učitele.

3.1.2 Slovní výukové metody

Vyprávění, rozhovor, napomínání a jiné slovní projevy se řadí mezi důležité edukační postupy, které jsou i v dnešní době nepostradatelné ve výchovně-vzdělávacím procesu. Pomocí verbálního projevu můžeme komunikovat, a tím pádem také přenášet informace. V dávných časech měla řeč podobu pouze v ústní formě, a pokusy je zachytit pomocí obrázků nebyly dostatečně přesné. Se vznikem písma, které už dokázalo přesněji a věrněji uchovat informace i na dlouhé vzdálenosti, a to jak prostorové, tak i časové, se lidská civilizace začala rychle rozvíjet (Maňák a Švec, 2003).

Výklad

Při aplikování této výukové metody, učitel předkládá žákům učivo postupně a po částech, čímž jej zprostředkovává žákům systematicky. Nestačí ovšem učivo žákům vykládat, je také třeba ověřovat, zda žáci určité části učiva pochopili. Jádrem výkladu je postup od nejpodstatnějších a konkrétních prvků učiva, až po abstraktní pojmy, od známých prvků

k neznámým a od jednoduchých ke složitým. Metoda výkladu se dá také kombinovat s jinými metodami, například společně s názorně-demonstrační, aby žáci viděli, jak se takové učivo může použít v praxi. Výhodou této metody je již zmíněná logická posloupnost, pomocí které se učivo dostává k žákům. Jako nevýhodu zde můžeme vidět nedostatek samostatného myšlení nebo komunikace ze strany žáků (Zormanová, 2012a).

Tato metoda lze využít například k představení a vysvětlení základních konceptů algoritmizace a programování, jako jsou datové typy, syntaxe apod. Učitel může využívat prezentace jednotlivých příkazů, ukázky kódu a jiné prvky k podpoření svého výkladu. Také lze výklad využít jako příležitost pro poskytnutí příkladů a aplikace jednotlivých příkazů v reálných situacích. To může žákům pomoci vysvětlovaným konceptům lépe porozumět.

Diskuse

Tato výuková metoda má podobu rozhovoru mezi všemi žáky a učitelem. Cílem diskuse je vyjasnění určitého problému, ať už jde o nové učivo nebo staré, které nebylo dostatečně osvojeno. Žáci by před zahájením diskuse měli vědět, že bude probíhat, a především na jaké téma, aby se na ni mohli patřičně připravit, nachystat otázky, argumenty apod. Pomocí této metody si žáci pouze neosvojují učivo, ale také si vyvíjejí komunikační kompetence. Učitel má za úkol diskusi pouze řídit a iniciovat žáky k vyjadřování svých nápadů a domněnek. Tato metoda se doporučuje především do vyšších ročníků základní školy, kde jsou žáci schopni sami za sebe argumentovat, rozvíjet své myšlenky a kriticky zhodnotit své názory a názory svých spolužáků (Skalková, 2007).

Tato metoda podporuje aktivní zapojení žáků, což umožňuje učiteli vést diskusi o základních konceptech algoritmizace a programování. Žáci mohou diskutovat o tom, jak jednotlivé koncepty fungují, jak se liší a kdy je vhodné je použít. Žáci mohou dále diskutovat o praktických aplikacích algoritmů a příkazů v reálném světě. To může zahrnovat diskusi o tom, jak jsou algoritmy používány ve všedním životě, v technologických inovacích nebo v různých odvětvích průmyslu.

3.1.3 Názorně-demonstrační výukové metody

Výše popsané slovní metody jsou ve výchovně-vzdělávacím prostředí nedílnou součástí. Nelze však využívat pouze slovní metody a poté od žáků očekávat, že budou schopni tyto vědomosti upotřebit v reálném světě. Tyto názorně-demonstrační metody vycházejí z principu názornosti, který zdůrazňuje důležitost názorného předvádění jevů a procesů

ve výuce. I v dnešní době dozajista platí Komenského Zlaté pravidlo *„Proto budiž učitelům zlatým pravidlem, aby všechno bylo předváděno všem smyslům, kolika možno. Totiž věci viditelné zraku, slyšitelné sluchu, vonné čichu, chutnatelné chuti a hmatatelné hmatu; a může-li něco být vnímáno najednou více smysly, budiž to předváděno více smyslům“* (Zormanová, 2012b). Názorně-demonstrační metody, stejně jako slovní, by se neměly používat izolovaně. Tyto dva typy metod se na sebe vážou, a je třeba je společně kombinovat (Maňák a Švec, 2003).

Předvádění a pozorování

Při aplikaci této výukové metody, žáci nejprve pozorují učitele při vykonávání, vysvětlování určitých předmětů a jevů. Následně se žáci sami snaží napodobovat učitele pomocí pokusů, které žákům smyslovým vnímáním zprostředkovávají prožitky, díky kterým mohou jevu či předmětu porozumět (Zormanová, 2012a).

Tato metoda je přínosná především tehdy, kdy učitel demonstruje konkrétní příkazy, postupy a řešení problémů. Učitel může žákům předvést, jak psát kód pro řešení určitého problému, nebo předvést nové příkazy, které pak žáci po učiteli opisují. Je také možné žákům předvést jakým způsobem odhalovat chyby v programu, a především jak tyto chyby řešit. Také lze demonstrovat různá vývojová prostředí, orientaci v nich apod.

Instruktáž

Instruktáž, jako názorně-demonstrační výuková metoda u žáků rozvíjí pohybové a pracovní dovednosti. Instruktáž může nabývat různých podob lze pojmout z více stran. Nejčastěji se používá slovní instruktáž doprovázená předvedením činnosti. Učitel tak vysvětluje žákům činnost verbálně a zároveň provádí činnost o které mluví, takže u žáků sledujících takovou instruktáž působí na více smyslů (Zormanová, 2012a).

Tato metoda je vhodná například při krokování algoritmů, kdy může učitel vést žáky krok za krokem přes proces tvorby a analýzy algoritmů a programů. Učitel může předvést žákům řešení konkrétních příkladů a cvičení, na kterých může demonstrovat použití jednotlivých příkazů. Pro lepší pochopení jednotlivých příkazů může učitel žáky odkázat na případné online návody nebo instruktážní videa.

3.1.4 Dovednostně-praktické výukové metody

Už v antice byla vyzdvihována filozofie, že škola by měla člověka připravit na život, a ne jej vzdělávat pouze v kontextu školy. V dnešní době se tato teze jen těžko splňuje,

jelikož znalostí, se kterými se žák musí seznámit v průběhu výchovně-vzdělávacího procesu je tolik, že se málo kdy najde prostor pro praktické uplatnění nabytých poznatků. Uplatňování takových poznatků se tedy odsouvá až do reálného světa, kde si žák musí vyzkoušet sám, zda si je dostatečně osvojil na to, aby je dokázal použít v praxi. Jedním z filozofů, který prosazoval praktické činnosti do škol, byl J. Dewey a jeho filozofie „learning by doing“ (učení se děláním) (Maňák a Švec, 2003).

Demonstračně-praktické metody se soustředí především na to, aby si žáci osvojili psychomotorické a motorické dovednosti. V dnešní době existuje koncepce tzv. činnostně orientované výuky. Při takové výuce je žákům umožněno přistupovat k učivu činnostmi. Některé předměty jsou celkově postavené na praktických činnostech, jako například informatika, a v některých se s propojováním učiva s reálnými experimenty setkáváme velmi často (fyzika, chemie). Ne ve všech vyučovacích předmětech jsou učitelé ochotni připravovat pro žáky praktické využití učiva, což žáky ochuzuje o možnost učit se efektivněji (Maňák a Švec, 2003). Podle pyramidy učení, kterou představil Edgar Dale, si pamatujeme 20 % z toho, co slyšíme, 30 % z toho, co vidíme, 70 % co říkáme a píšeme a 90 % z toho, co sami děláme (Cloke, 2023).

3.1.5 Organizační formy

Frontální vyučování v systému vyučovacích hodin

Frontální, nebo také hromadná výuka se vyznačuje prací s vymezenou skupinou žáků, kterou je většinou celá třída. Učitel pracuje s třídou v určitém čase po určitou dobu, čímž se zvyšuje jeho produktivita oproti individuální výuce, kde by měl učitel ve třídě pouze jednoho žáka. Učitel by měl mít na každou vyučovací hodinu připravený výukový cíl, který by měl s žáky naplnit. Dobře naformulovaný výukový cíl by měl být měřitelný (Kalhous a Obst, 2002). Při frontální výuce je důležitý osobní kontakt mezi učitelem a žáky, při kterém dochází k vzájemnému působení učitele a třídy. V průběhu výuky učitel nemusí konstantně komunikovat se třídou jako s celkem, ale má také prostor přistupovat k jednotlivcům, kteří potřebují pomoc při samostatných cvičeních. Učitel také může dbát na individuální potřeby jednotlivých žáků, ať už jde o speciální vzdělávací potřeby, různé styly učení nebo jejich emocionální stav (Skalková, 2007).

4 Metodické návrhy k výuce algoritmizace a programování

V této kapitole se zaměříme na tvorbu ucelené sady metodických návrhů zaměřených na problematiku JavaScriptu. Následující metodické návrhy jsou určeny především pro žáky, kteří již absolvovali výuku základů tvorby webových stránek, tzn. mají již znalosti týkající se jazyků, jako je HTML a CSS. Dále je třeba zmínit, že byť je JavaScript poměrně jednoduchým jazykem na osvojení, žáci základních škol by mohli mít problém s jeho syntaxí. Jedná se především o zápis hranatých `[]` a složených závorek `{}`. Žáci druhého stupně základních škol většinou nejsou vedeni k osvojení schopnosti psaní všemi deseti prsty, tudíž pro ně může být problém už jen samotné psaní na klávesnici, natož zápis speciálních znaků. Je tedy třeba zvážit, zda mají žáci dostatečné kompetence pro zvládnutí výuky JavaScriptu.

Metodické návrhy jsou koncipovány od nejjednodušších JavaScriptových prvků až po ty nejsložitější. Při čemž se v pozdější části také prolínají s jazykem HTML a jeho elementy. Výuka by měla probíhat tak, jak je popsána v sekci „Průběh výuky“, kde jsou také zmíněny důležité body, jako je prezentace nových příkazů, ukázka nových příkazů nebo samostatná práce. Metodické návrhy na sebe nemusí nutně navazovat z hodiny na hodinu, vlastně bychom doporučili mezi jednotlivé návrhy vložit hodinu na zopakování a procvičení jednotlivých příkazů, jelikož při nedostatečném upevnění probraných příkazů na ně mohou žáci rychle zapomenout a v dalších hodinách mohou mít problém s novými příkazy, které navazují na ty přechozí.

V prvním stádiu výuky jazyka JavaScript se žáci setkají pouze se samotným JavaScriptem bez jakýchkoliv dalších vazeb na ostatní jazyky. S jeho pomocí se naučí vypisovat text na obrazovku, buď ve formě vyskakovacích oken `alert` a `prompt`, nebo v konzoli. V dalších hodinách se výuka JavaScriptu bude ubírat směrem k propojení znalostí s jazyky HTML a CSS, kde si žáci vyzkoušejí na praktických příkladech, jakým způsobem JavaScript funguje na webových stránkách, a co všechno je třeba udělat před tím, než se obsah zobrazí na monitoru. Samostatné cvičení pro žáky jsou vymyšleny tak, aby v každé hodině upotřebili právě nabyté znalosti, a tak si je ještě více upevnili. Posledním metodickým návrhem je komplexní projekt „Hod kostkou“. Žáci by měli být motivováni při jeho vytváření tím, že ho budou moci využít v praxi, když budou chtít něco rozhodnout. V projektu totiž figurují dvě kostky, kde každá připadá jednomu hráči, a po stisknutí tlačítka program náhodně přiřadí čísla od jedné do šesti jednotlivým hráčům. Žáci tedy mohou program využít například místo hry „kámen, nůžky, papír“ při domluvě se spolužáky, kamarády či rodiči.

4.1 Metodický návrh č. 1 – Úvod do jazyka JavaScript

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím příkazů alert() a prompt(), prostřednictvím kterého vypíše celé své jméno.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, kontrola.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, diskuse, instruktáž.

Doporučené didaktické zásady:

- Přiměřenosti, spojení teorie s praxí.

Pomůcky:

- Počítače s připojením k internetu, projektor.

Časová dotace:

- 45 minut.

Úvodní otázky pro žáky:

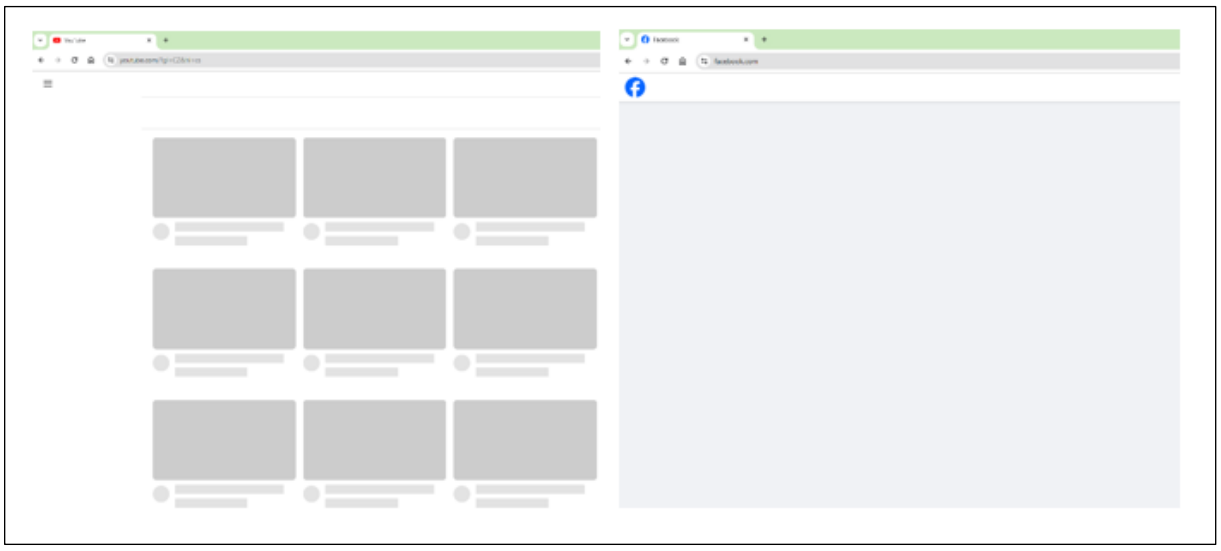
- Jaké jazyky používáme k tvorbě webových stránek?
- Slyšeli jste už někdy o jazyce JavaScript?
- K čemu by se mohl JavaScript používat?

Průběh výuky:

1. Představení výukového cíle žákům.

2. Kladení úvodních otázek žákům.
3. Diskuse s žáky ohledně fungování webových stránek.
4. Prezentování JavaScriptu žákům.
5. Zprovoznění webové konzole.
6. Představení základních příkazů JavaScriptu.
7. Žáci pracují podle pokynů učitele.
8. Žáci pracují na samostatné činnosti.
9. Prezentace výsledků a zhodnocení práce.
10. Zhodnocení samostatné práce žáky.

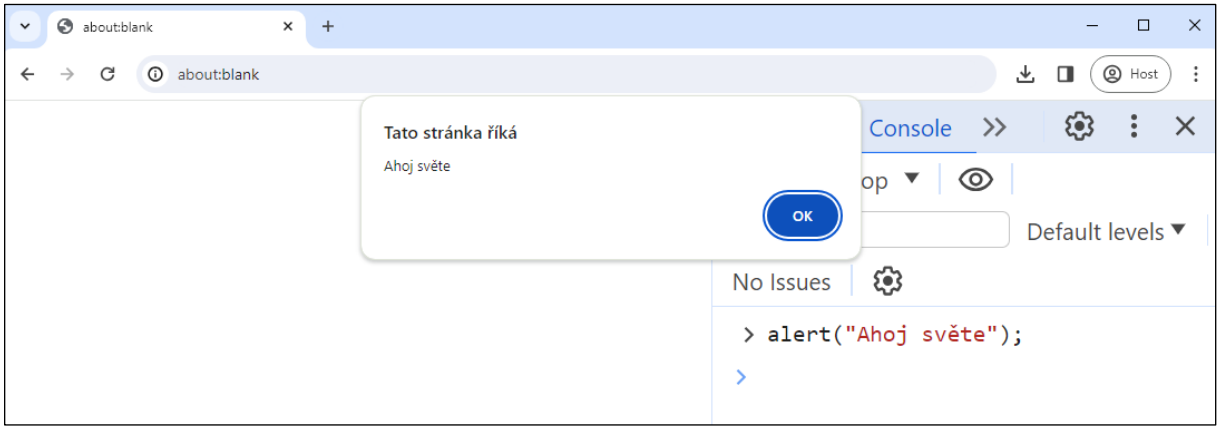
Cílem této hodiny je nechat si žáky vyzkoušet, jak fungují základní příkazy v jazyce JavaScript. Konkrétně jde o příkazy `alert()` a `prompt()`. Nejprve je potřeba žákům připomenout formou diskuse, jak vypadají webové stránky, jakým způsobem fungují a jaké jazyky se většinou využívají k jejich vytvoření. Po krátké diskusi učitel žákům prezentuje programovací jazyk JavaScript, jako jazyk, který přináší webovým stránkám funkčnost. Učitel může žákům předvést, jak na webovém prohlížeči Google Chrome zakázat používaní JavaScriptu (**Nastavení – Ochrana soukromí a zabezpečení – Nastavení webu – JavaScript – Nepovolovat webům používat JavaScript**) a ukázat jim na vybraných webových stránkách (YouTube, Facebook, Centrum, …) jak je JavaScript v internetovém světě důležitý (viz Obrázek 1).



Obrázek 1: YouTube a Facebook bez JavaScriptu

(Zdroj: vlastní zpracování)

Dalším úkolem je otevření webové konzole JavaScriptu přímo v prohlížeči Google Chrome (klávesová zkratka na OS Windows „**Ctrl + Shift + I**“ nebo **Další nástroje – Nástroje pro vývojáře**). Zde může učitel žákům předvést první JavaScriptový příkaz, kterým je `alert()`. Pomocí tohoto příkazu si mohou žáci nechat vypsat jakoukoliv zprávu, kterou napíšou do závorky do uvozovek (viz Obrázek 2).



Obrázek 2: Příklad použití příkazu alert()

(Zdroj: vlastní zpracování)

Podobným způsobem funguje také příkaz `prompt()`. Učitel předvede žákům příklad použití tohoto příkazu, který přijímá uživatelův text (input). Při zkombinování těchto dvou příkazů mohou žáci zadat text přímo do vyskakovacího pole `prompt()` a následně si jej nechat zobrazit pomocí příkazu `alert()`. To vše lze ještě propojit pomocí znaménka `+`, které lze vložit mezi příkaz `prompt()` a libovolný textový řetězec (viz Obrázek 3).

Následuje samostatná práce, kde si žáci sami vyzkoušejí prezentované příklady použití jednotlivých příkazů.

Návrh pro téma samostatné práce:

- Napiš své jméno do příkazu `alert()` a pomocí příkazu `prompt()` si nechej vypsat své celé jméno i s příjmením.

```
alert("František " + prompt("Napiš své příjmení: "));
```

Obrázek 3: Příklad použití příkazu prompt()

(Zdroj: vlastní zpracování)

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.2 Metodický návrh č. 2 – Proměnné

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím proměnných, prostřednictvím kterého vypíše požadovanou zprávu.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Znalost JavaScriptových příkazů alert() a prompt().

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Přiměřenosti, spojení teorie s praxí.

Pomůcky:

- Počítače s připojením k internetu, projektor.

Časová dotace:

- 45 minut.

Úvodní otázky pro žáky:

- Víte, co je to vývojové prostředí?
- Slyšeli jste už někdy o proměnných?
- K čemu by mohly proměnné sloužit?

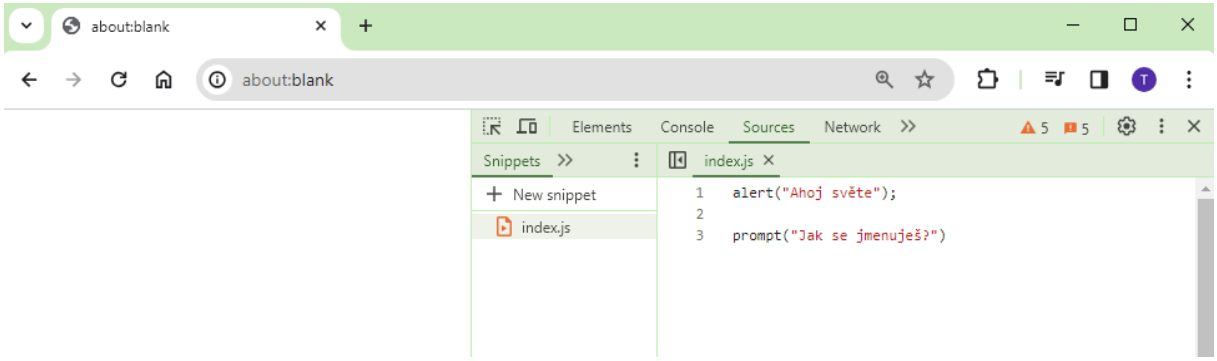
Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování příkazů z předchozí hodiny.
5. Prezentování proměnných žákům.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatné činnosti.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

Cílem této hodiny je seznámit žáky s proměnnými v jazyce JavaScript a naučit je základní práci s nimi. Nejprve je potřeba otevřít si patřičné vývojové prostředí, jelikož ve webové konzoly se komplexnější kód tvořit nedá. Respektive dá, ale je to zbytečně složité, když existuje spoustu jiných, k tomu určených programů. Existuje několik možností pro výběr vývojového prostředí:

1. Pokud nechceme, aby se žáci museli pokaždé přihlašovat (žáci často zapomínají nebo ztrácejí přihlašovací údaje), lze využít v prohlížeči Google Chrome přímo zabudované vývojové prostředí **snippets**, které najdeme v záložce **Sources** v **Nástrojích pro vývojáře**, kde si lze vytvořit své vlastní soubory a pracovat s nimi přímo v prohlížeči (viz Obrázek 4).

Další možností bez registrace je online vývojové prostředí **JavaScript Playground** na adrese https://playcode.io/ (viz Obrázek 5).



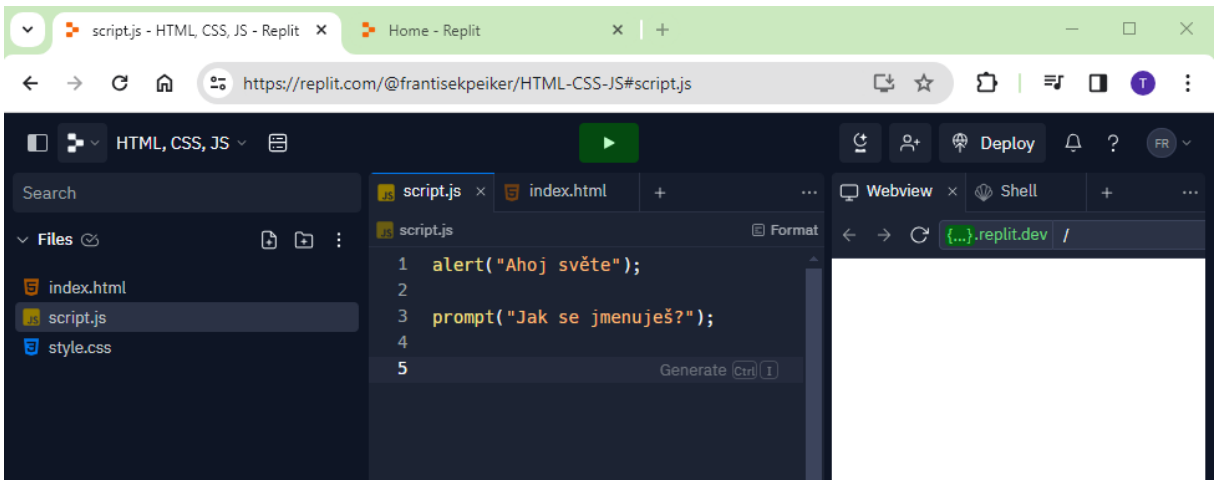
Obrázek 4: Snippets

(Zdroj: vlastní zpracování)

Obrázek 5: JavaScript Playground

(Zdroj: vlastní zpracování)

2. Další možností, kde je už ale potřeba registrace, je online vývojové prostředí **Replit** na adrese https://replit.com/, které je mimo jiné obohacené i o umělou inteligenci, což může žákům výrazně zjednodušit začátky v programování (viz Obrázek 6).



Obrázek 6: Replit

(Zdroj: vlastní zpracování)

3. Také se ovšem nabízí možnost využít některého z vývojových prostředí v lokálním prostředí, jako například **Atom**, **Visual Studio Code** nebo **WebStorm**. Při výběru některého z těchto vývojových prostředí je však potřeba je nejprve žákům nainstalovat. Proto se, minimálně do začátku, doporučují online vývojová prostředí.

Ve vybraném vývojovém prostředí je možné ukázat žákům příkazy, které se již v minulé hodině naučili, a zopakovat s nimi jejich funkce, a kdy se který používá. Dále je možné žákům představit proměnné. Je několik možností, kterými lze proměnnou zapsat:

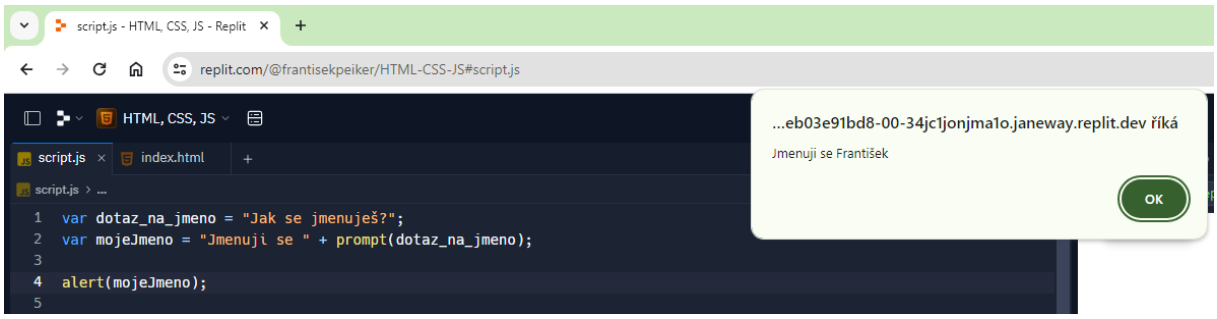
1. Pomocí klíčového slova `var` – nejjednodušší na vysvětlení, `var` je zkratka z anglického variable (proměnná), tudíž nejvhodnější na používání v začátcích programování.
2. Pomocí klíčového slova `let` – má téměř shodnou funkčnost jako `var`.
3. Pomocí klíčového slova `const` – příliš se nedoporučuje. Je to zkratka z anglického constant (konstantní). Již z názvu lze odhadnout, že když do proměnné zapíšeme hodnotu, dále už tato hodnota nepůjde přepsat.

Učitel žákům ukazuje způsob použití proměnné na tom, což již znají, tzn. příkazy `alert()` a `prompt()` (viz Obrázek 7). Je také třeba žákům vysvětlit, jak proměnné uchovávají hodnoty, a že existují určitá pravidla pro jejich pojmenovávání.

Obrázek 7: Příklad použití proměnné

(Zdroj: vlastní zpracování)

Dále učitel ukazuje žákům, že lze vytvořit více proměnných (viz Obrázek 8). Žáci opisují příklad podle učitele. Poté žáci pracují na samostatné činnosti.



Obrázek 8: Příklad použití více proměnných a jejich názvů

(Zdroj: vlastní zpracování)

Návrh pro téma samostatné práce:

- Pomocí několika proměnných si nechej vypsat své jméno, příjmení a věk.

```
var dotaz_na_jmeno = "Jaké je tvé celé jméno?";
var mojeJmeno = "Jmenuji se " + prompt(dotaz_na_jmeno);
var dotaz_na_vek = "Kolik je ti let?";
var mujVek = " a je mi " + prompt(dotaz_na_vek) + " let";

alert(mojeJmeno + mujVek);
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

Poznámka:

Samostatná práce nemusí být nutně zaměřena na vypsání jména a věku, žáci mohou napsat svého oblíbeného fotbalového/basketbalového hráče, svého domácího mazlíčka a jeho jméno apod.

4.3 Metodický návrh č. 3 – Datové typy a konzola

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím základních matematických operací, prostřednictvím kterého vypíše výsledky jednotlivých operací do konzole.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, kontrola, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Znalost proměnných v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Přiměřenosti, spojení teorie s praxí, uvědomělosti a aktivity.

Pomůcky:

- Počítače s připojením k internetu, projektor.

Časová dotace:

- 45 minut.

Úvodní otázky pro žáky:

- Slyšeli jste už někdy o datových typech?
- Kolik datových typů by asi mohlo existovat?
- Slyšeli jste už někdy o konzoli?

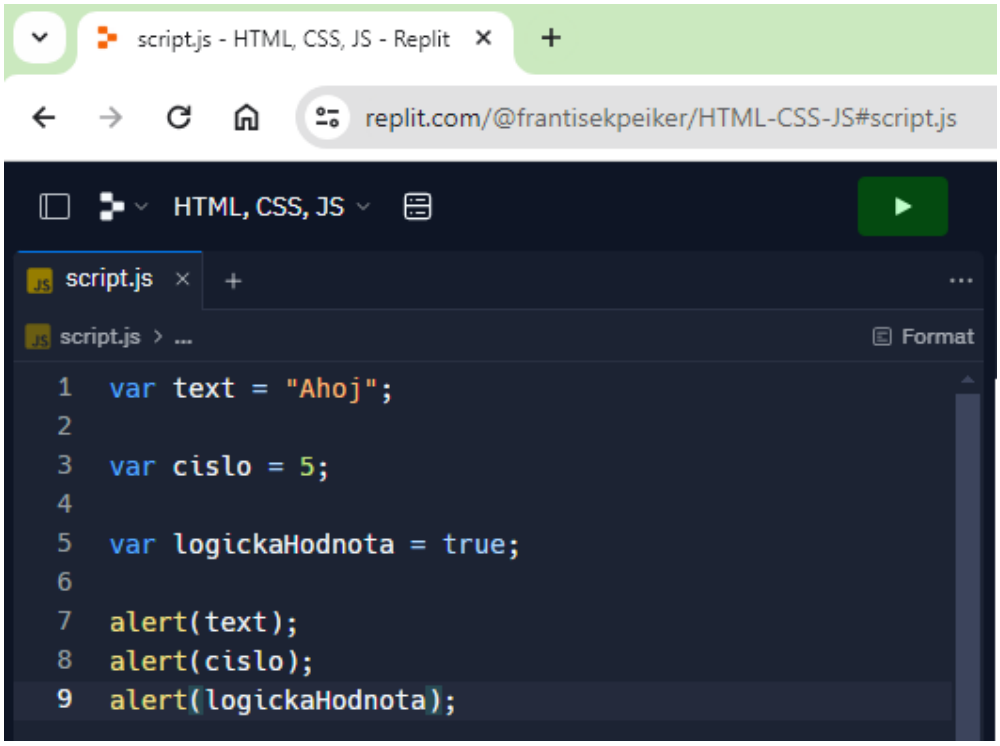
Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování zápisu proměnných.
5. Prezentování různých typů proměnných žákům.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatném cvičení.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

Cílem této hodiny je seznámit žáky s různými datovými typy a procvičení proměnných. Nejprve je vhodné s žáky prodiskutovat, co si pamatují z poslední hodiny, jakým klíčovým slovem se označuje proměnná apod. Učitel spolu s žáky otevřou vybrané vývojové prostředí, ve kterém si mohou zápis proměnných zopakovat (Replit, JavaScript Playground, …). Učitel dále vysvětluje na příkladech základní typy proměnných:

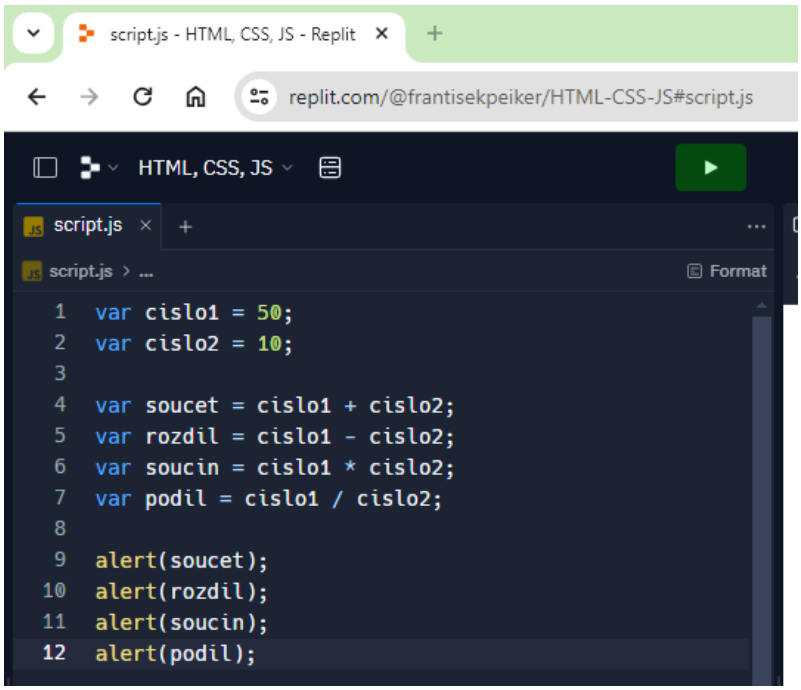
1. Textový řetězec– jde o text, který se píše mezi uvozovky.
2. Číslo – jde o číselné hodnoty, které se nezapisují do uvozovek.
3. Boolean (pravdivostní hodnota) – jde o hodnotu `true` nebo `false` (pravda nebo nepravda). Tato hodnota se stejně jako číslo nezapisuje do uvozovek.

Žáci opisují příklady podle učitele (viz Obrázek 9). Po seznámení se s různými datovými typy učitel prezentuje, jakými znaménky se značí jednotlivé matematické operace. Žáci napodobují učitelovi příklady (viz Obrázek 10).



Obrázek 9: Příklad různých datových typů

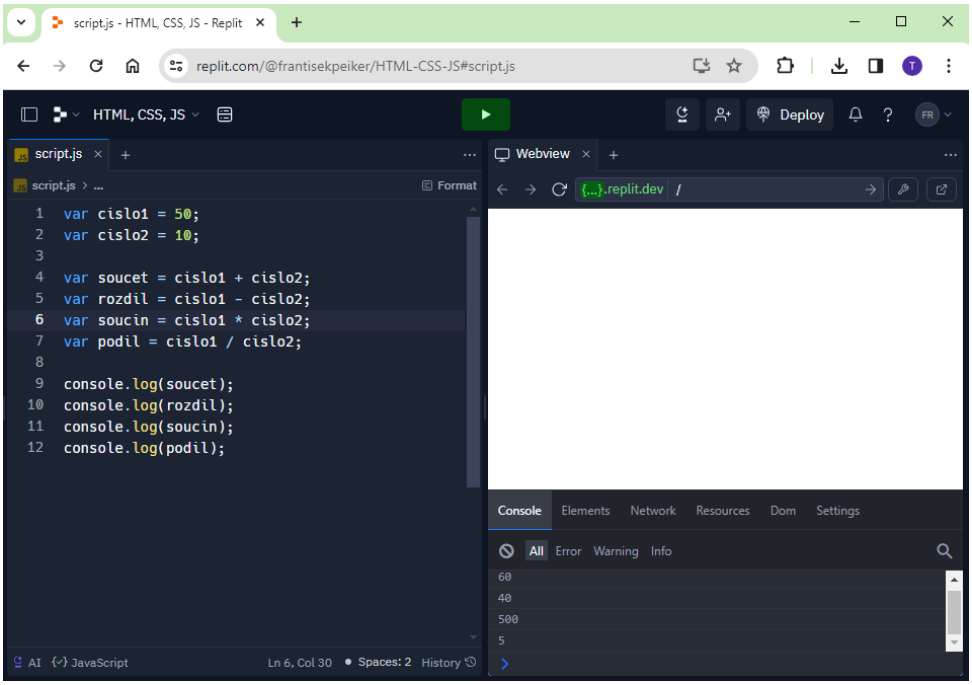
(Zdroj: vlastní zpracování)



Obrázek 10: Příklad různých matematických operací

(Zdroj: vlastní zpracování)

Jelikož se v ukázce zobrazuje několikrát za sebou příkaz `alert()` a může to být rušivé, možná až otravné jej vždycky odkliknout, učitel sdělí žákům, že existuje další způsob jak si nechat vypsat text, a tím je příkaz `console.log()`, který vypíše požadovanou zprávu do konzole (viz Obrázek 11). Po dostatečném procvičení mohou žáci pracovat na samostatné práci.



Obrázek 11: Příklad použití příkazů console.log()

(Zdroj: vlastní zpracování)

Návrh pro téma samostatné práce:

- Dopište před výsledek matematické operace text, aby bylo jasné, o kterou matematickou operaci se jedná.

```
var cislo1 = 50;
var cislo2 = 10;

var soucet = cislo1 + cislo2;
var rozdil = cislo1 - cislo2;
var soucin = cislo1 * cislo2;
var podil = cislo1 / cislo2;

console.log("Součet čísel " + cislo1 + " a " + cislo2 + " je " +
soucet);
console.log("Rozdíl čísel " + cislo1 + " a " + cislo2 + " je " +
rozdil);
console.log("Součin čísel " + cislo1 + " a " + cislo2 + " je " +
soucin);
console.log("Podíl čísel " + cislo1 + " a " + cislo2 + " je " + podil);
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.4 Metodický návrh č. 4 – Funkce

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím funkcí, prostřednictvím kterého vypíše požadovaný text.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Znalost proměnných v jazyce JavaScript a příkazů alert() a prompt().

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, diskuse, instruktáž.

Doporučené didaktické zásady:

- Přiměřenosti, spojení teorie s praxí, soustavnosti.

Pomůcky:

- Počítače s připojením k internetu, projektor.

Časová dotace:

- 45 minut.

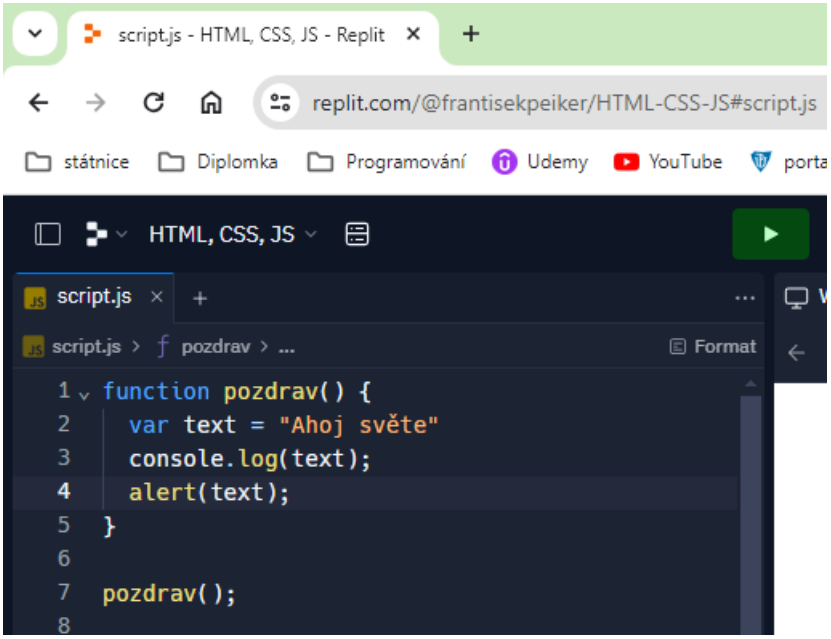
Úvodní otázky pro žáky:

- Slyšeli jste už někdy o funkcích?
- Jaké výhody může mít funkce?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Opakování proměnných.
5. Prezentování principu funkce žákům.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatném cvičení.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

Cílem těchto dvou hodin je naučit žáky používat funkce pro zjednodušení práce. Nejprve je potřeba otevřít vybrané vývojové prostředí, kde učitel společně s žáky zopakují datové typy a zápis proměnných. Učitel dále prezentuje nové učivo, kterým jsou JavaScriptové funkce. Funkce se podobně jako proměnná musí deklarovat klíčovým slovem. V případě funkcí je takové klíčové slovo `function`, za které následuje název funkce, jednoduché závorky a složené závorky (viz Obrázek 12). Do jednoduchých závorek `()` zatím není potřeba nic psát, ale je vhodné žákům vysvětlit, že se do nich zapisují tzv. parametry funkce, se kterými funkce



Obrázek 12: Příklad použití funkce

(Zdroj: vlastní zpracování)

dále pracuje. Do složených závorek `{}` se zapisuje kód funkce, totiž to, co chceme, aby funkce provedla. Důležité je také vysvětlit žákům, že se funkce provede až tehdy, kdy je zavolána.

Učitel vysvětluje žákům na příkladu výhody funkce, jako možnost spuštění větší části kódu, než by bylo možné v proměnné, spuštění bloku kódu až když je potřeba apod. Žáci přepisují učitelův zápis funkce s drobnými úpravami, aby se s funkcemi lépe seznámili. Po dostatečném procvičení zápisu funkcí mohou žáci přejít k práci na samostatném cvičení.

Návrh pro téma samostatné práce:

- Do těla funkce napiš program, pomocí kterého necháš uživatele zadat své jméno a následně zadané jméno vypiš i s pozdravem.

```
function pozdrav() {
  var jmeno = prompt("Zadej své jméno: ");
  alert("Ahoj, " + jmeno + "!");
}

pozdrav();
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.5 Metodický návrh č. 5 – Práce s funkcemi

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím funkci, prostřednictvím kterého vypíše požadovaný text na obrazovku.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.
- Znalost proměnných a funkcí v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Trvalosti, spojení teorie s praxí, soustavnosti.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut.

Úvodní otázky pro žáky:

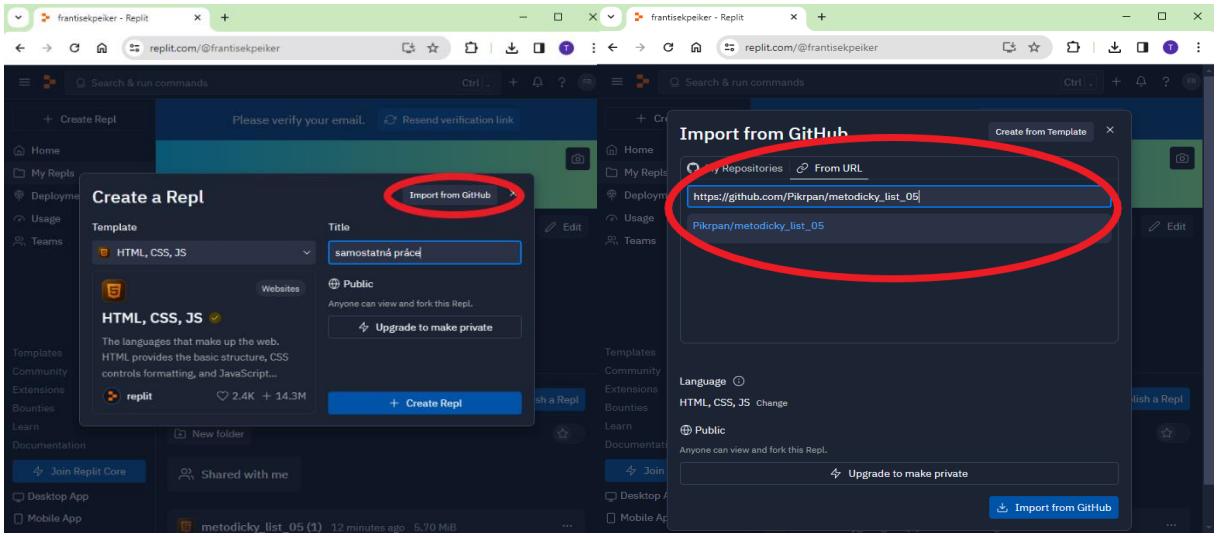
- Jakým klíčovým slovem se deklaruje funkce?
- Kdy se provede program zapsaný ve funkci?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování zápisu funkcí.
5. Žáci pracují na samostatném cvičení (pracovní list).
6. Prezentace výsledků a zhodnocení práce.
7. Zhodnocení samostatné práce žáky.

Náplní hodiny je převážně samostatná práce žáků na cvičení (pracovním listu). Nejprve žáci spolu s učitelem otevřou vývojové prostředí, ideálně **Replit**, nebo některé z lokálních prostředí, jako jsou například **Atom**, **Visual Studio Code** nebo **WebStorm**. Učitel s žáky zopakuje syntaxi funkcí a způsob jejich fungování.

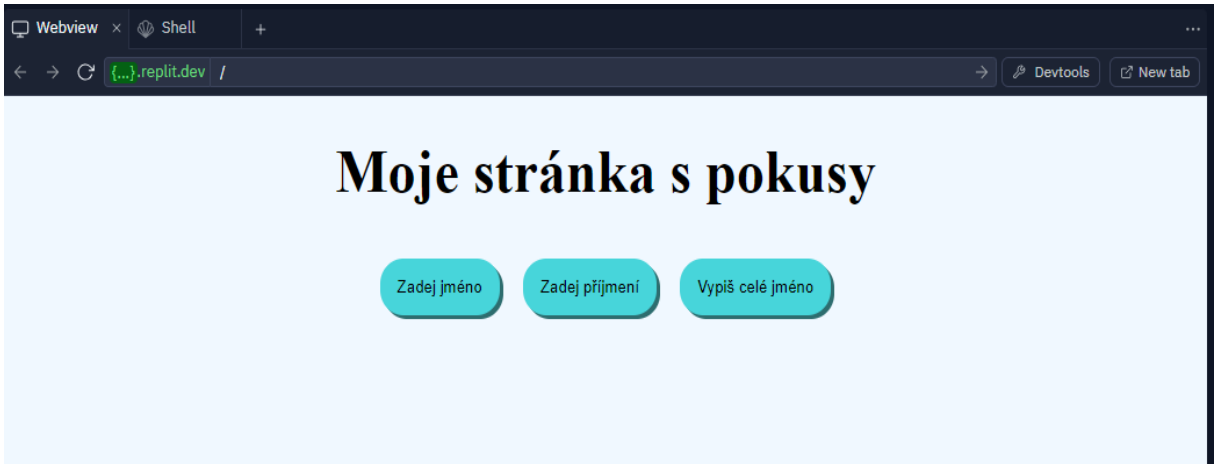
Pro samostatnou práci žáků je třeba buď stáhnout soubory z webové stránky GitHub na adrese https://github.com/Pikrpan/metodicky_list_05, nebo v případě používaní vývojového prostředí **Replit** stačí zadat odkaz do kolonky **Import from GitHub** (viz Obrázek 13).



Obrázek 13: Vložení odkazu pro GitHub projekt

(Zdroj: vlastní zpracování)

Projekt obsahuje soubory index.html, styles,css a main.js. V souborech index.html a styles.css není třeba cokoliv upravovat. JavaScriptový soubor main.js obsahuje pouze komentáře s instrukcemi pro žáky. Před zahájením samostatné práce učitel vysvětluje žákům, že jsou na stránce zobrazeny tři tlačítka, a každé z nich po stisknutí volá jinou funkci, tudíž po deklarování funkce ji žáci nemusí volat znovu (viz Obrázek 14). Žáci tedy upravují pouze JavaScriptový soubor main.js podle instrukcí.



Obrázek 14: Vzhled stránky samostatného cvičení

(Zdroj: vlastní zpracování)

Žáky upravený soubor by mohl vypadat například následovně:

```
//Zde vytvoř proměnné jmeno a prijmeni
var jmeno, prijmeni;

//Zde vytvoř funkci s názvem zadejJmeno
function zadejJmeno(){
    //V těle funkce zadejJmeno přiřaď proměnné jmeno hodnotu prompt()
    jmeno = prompt("Zadej své jméno: ");
}

//Zde vytvoř funkci s názvem zadejPrijmeni
function zadejPrijmeni(){
    //V těle funkce zadejPrijmeni přiřaď proměnné prijmeni hodnotu
prompt()
    prijmeni = prompt("Zadej své příjmení: ");
}

//Zde vytvoř funkci s názvem vypisCeleJmeno
function vypisCeleJmeno(){
    //V těle funkce vypisCeleJmeno vypiš hodnoty jmeno a prijmeni
pomocí příkazu alert()
    alert("Mé jméno je " + jmeno + " " + prijmeni + ".");
}
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.6 Metodický návrh č. 6 – Úvod k DOM (Objektový Model Dokumentu)

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím příkazů pro práci s DOM, prostřednictvím kterého změní vzhled webové stránky.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.
- Znalost proměnných v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Přiměřenosti, uvědomělosti a aktivity, spojení teorie s praxí, soustavnosti.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut.

Úvodní otázky pro žáky:

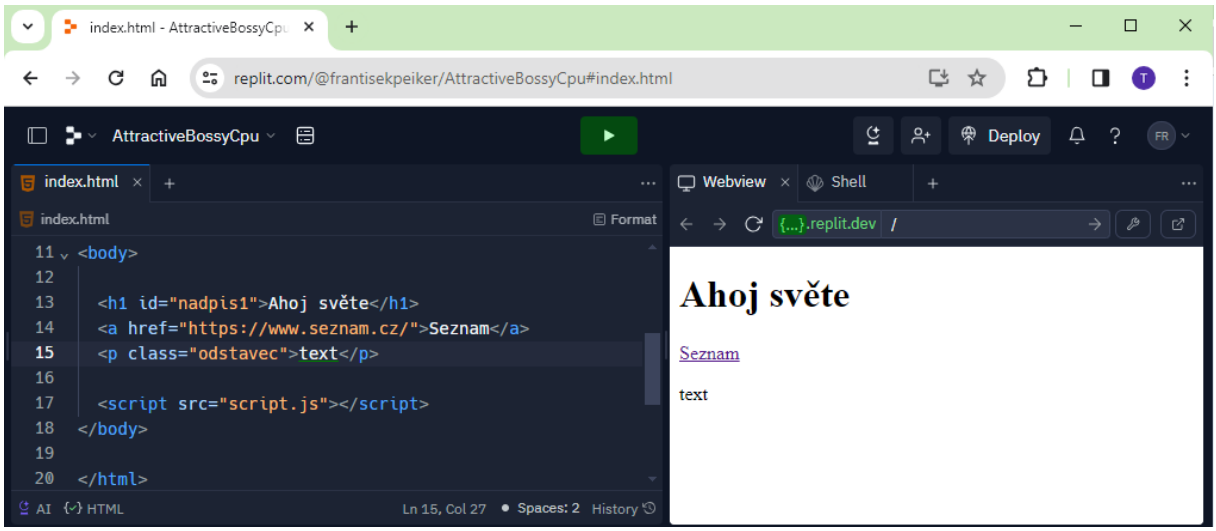
- Znáte nějaké názvy HTML elementů?
- Víte, co jsou to atributy HTML elementů?
- K čemu by mohl sloužit id atribut?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Připomenutí základních atributů ve značkovacím jazyce HTML.

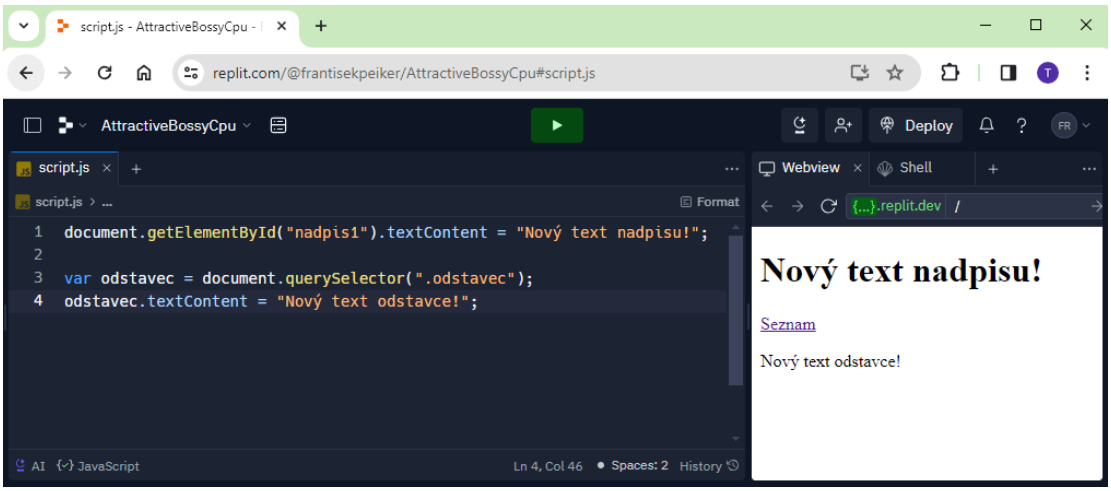
5. Představení nových příkazů.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatné práci.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

Cílem hodiny je seznámit s žáky s novými příkazy `document.querySelector()`, `document.getElementById()` a `.textContent`. Příkazy se používají, pokud je třeba změnit obsah textu u jednotlivých HTML elementů. Nejprve učitel společně s žáky otevřou vývojové prostředí a vytvoří nový dokument podle předlohy HTML, CSS a JavaScriptu. Učitel žákům připomíná jednotlivé HTML elementy a jejich atributy jako id, src, href, class apod. Důležité je, aby učitel žákům předvedl, kam se atributy elementů zapisují, a kde je mají žáci hledat. Žáci si zápis různých atributů zkoušejí podle pokynů učitele (viz Obrázek 15).



Obrázek 15: Příklad ukázky HTML elementů a jejich atributů
(Zdroj: vlastní zpracování)

Po patřičném zopakování základního zápisu HTML elementů a jejich atributů učitel prezentuje žákům nové příkazy v jazyce JavaScript. Učitel vysvětluje, kdy a proč se tyto příkazy používají a předvádí je na názorných příkladech (viz Obrázek 16).

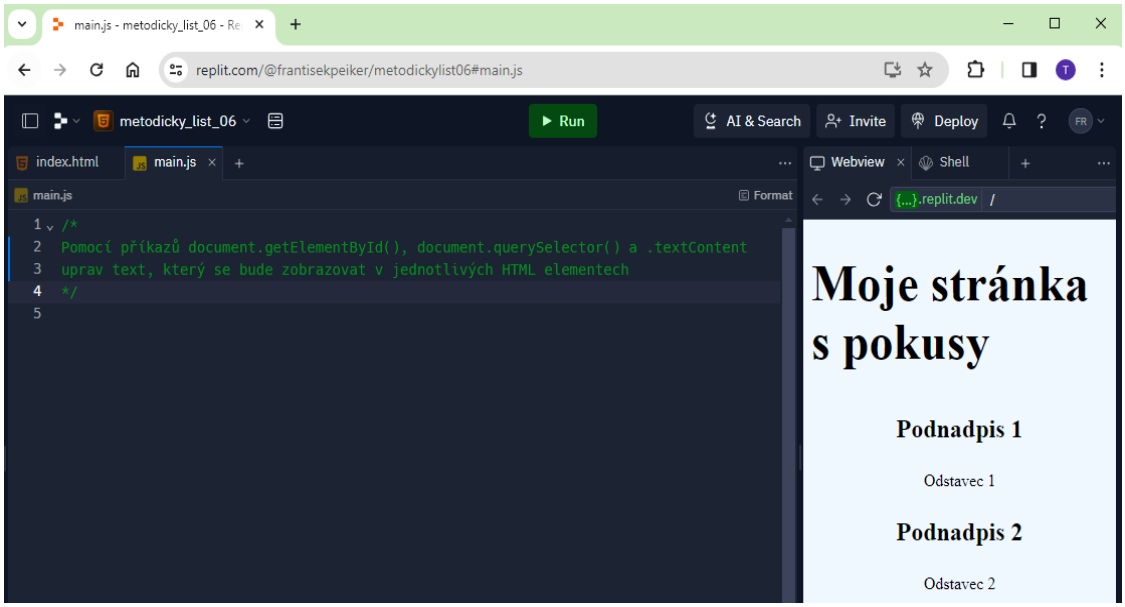


Obrázek 16: Příklad ukázky příkazů .getElementById(), .querySelector() a .textContent

(Zdroj: vlastní zpracování)

Po procvičení nových příkazů mohou žáci začít pracovat na samostatné práci, kterou je projekt umístěn na stránce GitHub na adrese https://github.com/Pikrpan/metodicky_list_06. Projekt lze stáhnout v případě používání lokálního vývojového prostředí nebo jej lze přímo importovat do vývojového prostředí Replit přes uvedený URL odkaz. V samostatném projektu žáci upravují text u jednotlivých, již vytvořených HTML elementů. Žáci musejí správně rozhodnout, který z nových příkazů použít u různých elementů (viz Obrázek 17).

Žáky upravený soubor by mohl vypadat například takto:



Obrázek 17: Původní vzhled stránky samostatné práce

(Zdroj: vlastní zpracování)

```
/*
Pomocí příkazů getElementById a querySelector uprav text,
který se bude zobrazovat v jednotlivých elementech
*/

var podnadpis1 = document.getElementById("podnadpis1");
podnadpis1.textContent = "Mé jméno:";

var odstavec1 = document.querySelector("p");
odstavec1.textContent = "František Peiker";

var podnadpis2 = document.getElementById("podnadpis2");
podnadpis2.textContent = "Můj věk:"

var odstavec2 = document.getElementById("odstavec2");
odstavec2.textContent = "25";

var podnadpis3 = document.getElementById("podnadpis3");
podnadpis3.textContent = "Moje bydliště:"

var odstavec3 = document.querySelector("h3");
odstavec3.textContent = "Stěbořice";
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.7 Metodický návrh č. 7 – Pole

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím příkazů pro práci s DOM, prostřednictvím kterého aktualizuje HTML elementy na webové stránce.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.
- Znalost funkcí a proměnných v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Přiměřenosti, spojení teorie s praxí, trvalosti.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut.

Úvodní otázky pro žáky:

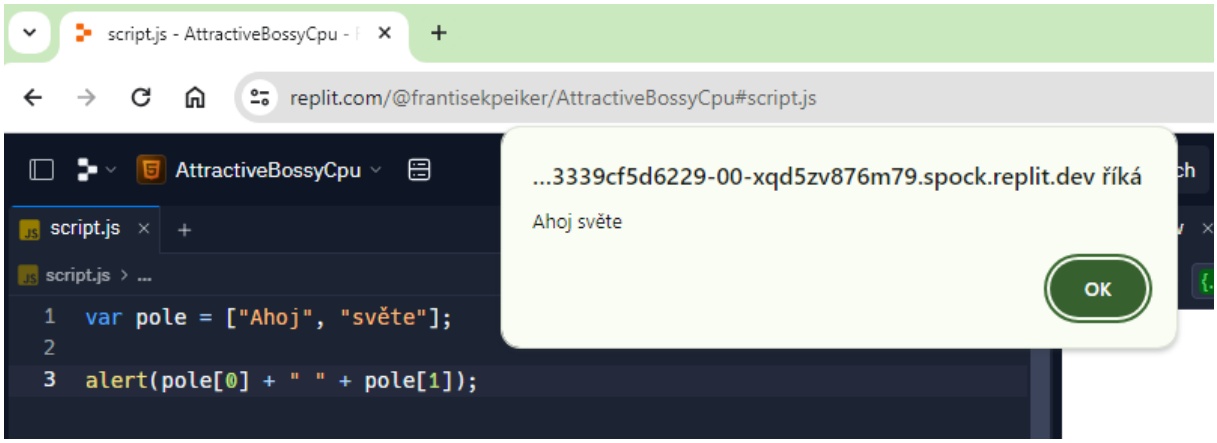
- Víte, co je to pole v programování?
- K čemu by mohly pole sloužit?
- Jaké další atributy kromě id znáte?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování učiva předchozí hodiny.
5. Představení JavaScriptových polí.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatné práci.

8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

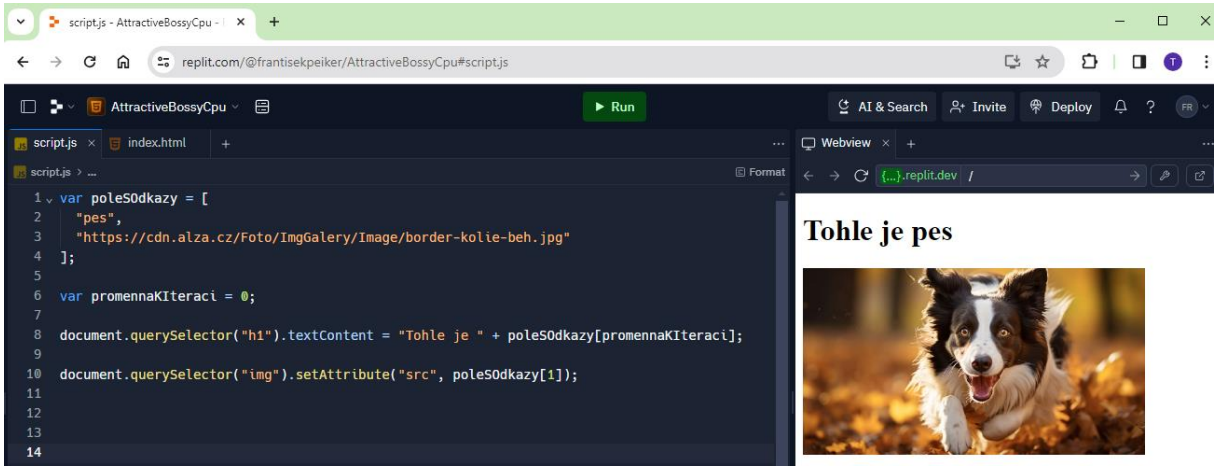
Cílem této hodiny je seznámit žáky s JavaScriptovým polem a naučit je základní práci s ním. Nejprve je třeba zopakovat předchozí příkazy `document.querySelector()`, `document.getElementById()` a `.textContent`, jelikož je v samostatné práci budou žáci potřebovat. Učitel společně s žáky otevřou vybrané vývojové prostředí a procvičují v něm naučené příkazy. Po zopakování předchozího učiva učitel představuje žákům nové, kterým je JavaScriptové pole. Pole se stejně jako proměnná značí klíčovým slovem `var`. Je zde však jeden rozdíl, a to sice takový, že do pole lze zapsat více hodnot. Tyto hodnoty se zapisují mezi hranaté závorky a oddělují se od sebe čárkou. Tyto hodnoty se později dají jednotlivě volat. Je třeba vysvětlit žákům, že se hodnoty v poli počítají od nuly, nikoliv od jedničky. Tudíž první hodnota v poli má index 0, druhá hodnota 1, atd. (viz Obrázek 18).



Obrázek 18: Ukázka příkladu zápisu pole v jazyce JavaScript

(Zdroj: vlastní zpracování)

Dále učitel představuje žákům další příkaz, pomocí kterého lze manipulovat s DOM (Objektový Model Dokumentu). Tímto příkazem je `.setAtribute()`, pomocí kterého lze přiřadit jednotlivým HTML elementům různé atributy. Je zde však rozdíl oproti příkazu `.textContent`, a to takový, že v případě příkazu `.setAtribute()` se hodnoty, které se mají změnit, zapisují obě do jednoduchých závorek a oddělují se od sebe čárkou (viz Obrázek 19).



Obrázek 19: Příklad použití příkazu .setAtribute() a proměnné určené k iteraci

(Zdroj: vlastní zpracování)

Po procvičení nových příkazů mohou žáci začít pracovat na samostatné práci, kterou je projekt umístěn na stránce GitHub na adrese https://github.com/Pikrpan/metodicky_list_07. Projekt lze stáhnout v případě používání lokálního vývojového prostředí nebo jej přímo importovat do vývojového prostředí Replit přes uvedený URL odkaz. V samostatném projektu žáci upravují pouze JavaScriptový soubor main.js a to podle pokynů v komentářích.

Žáky upravený soubor by mohl vypadat například následovně:

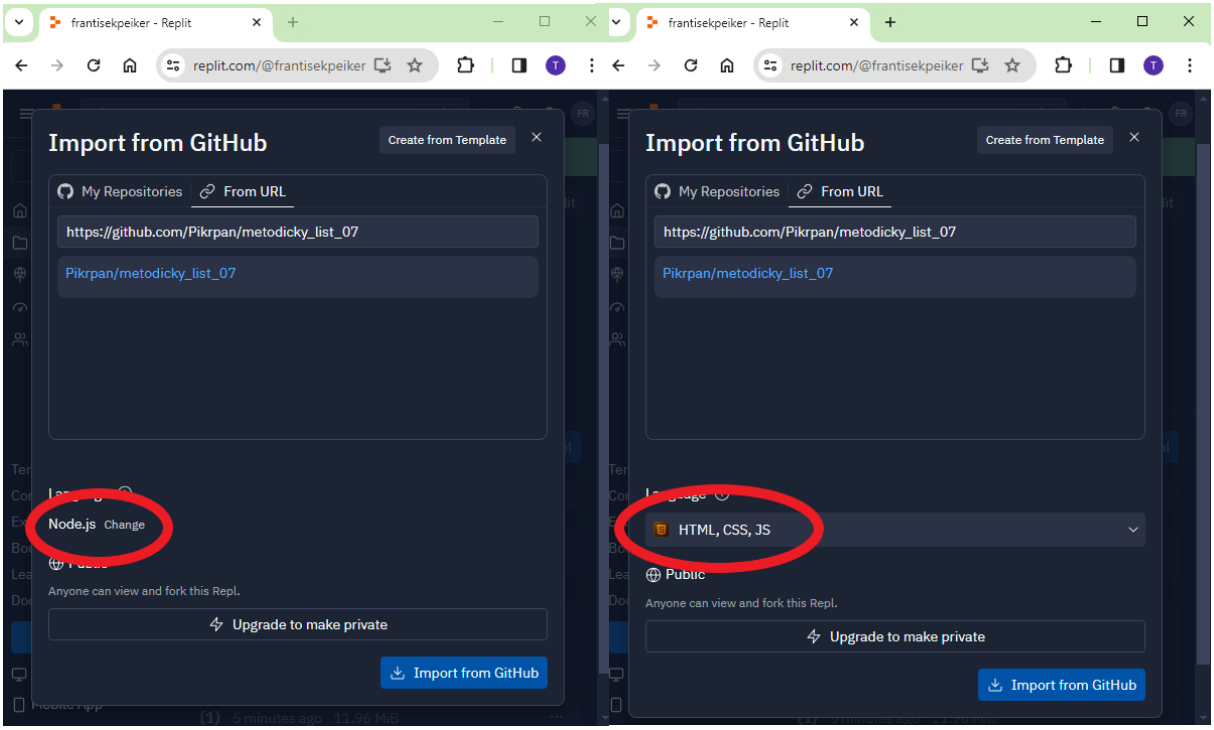
```
var poleZvirat = ["pes", "kočka", "křeček"];

var imgSrc = [
    "https://cdn.alza.cz/Foto/ImgGalery/Image/border-kolie-beh.jpg",
    "https://www.chlupaci.cz/user/articles/images/kocici-plemeno-
kocka-domaci_(11).jpg",
    "https://www.prokrecka.cz/wp-
content/uploads/2021/02/krecek_syrsky_zlaty_1.jpg"
];

//Vytvoř proměnnou tak, aby se odkazovala na HTML element h2
var h2 = document.querySelector("h2");

//Vytvoř proměnnou tak, aby se odkazovala na HTML element img
var img = document.getElementById("zvireImg");
```

```
//Vytvoř proměnnou, která bude sloužit k iteraci polí
var i = 0;

//Změň text v h2 elemntu na "Tohle je " + aktuální zvíře
h2.textContent = "Tohle je " + poleZvirat[i];

//Změň obrázek tak, aby souhlasil s textem h2 elementu (pomocí
.setAtribute() příkazu)
img.setAttribute("src", imgSrc[i]);

function dalsiZvire(){
    //Přičti k proměnné určené k iteraci + 1
    i++;

    //Změň text v h2 elementu a obrázek podle proměnné určené k iteraci
    h2.textContent = "Tohle je " + poleZvirat[i];
    img.setAttribute("src", imgSrc[i]);
}
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

Poznámka:

- Může nastat problém při importování souborů z GitHubu, který automaticky vybere Node.js jako předlohu (Template) pro projekt. Tento problém lze jednoduše vyřešit změněním předlohy na HTML, CSS a JS (viz Obrázek 20).



Obrázek 20: Řešení možného problému

(Zdroj: vlastní zpracování)

4.8 Metodický návrh č. 8 – Kondicionály

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím podmínek a příkazů pro práci s DOM, prostřednictvím kterého aktualizuje HTML elementy na webové stránce.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.
- Znalost proměnných a funkcí v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Soustavnosti, spojení teorie s praxí, trvalosti, uvědomělosti a aktivity.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut

Úvodní otázky pro žáky:

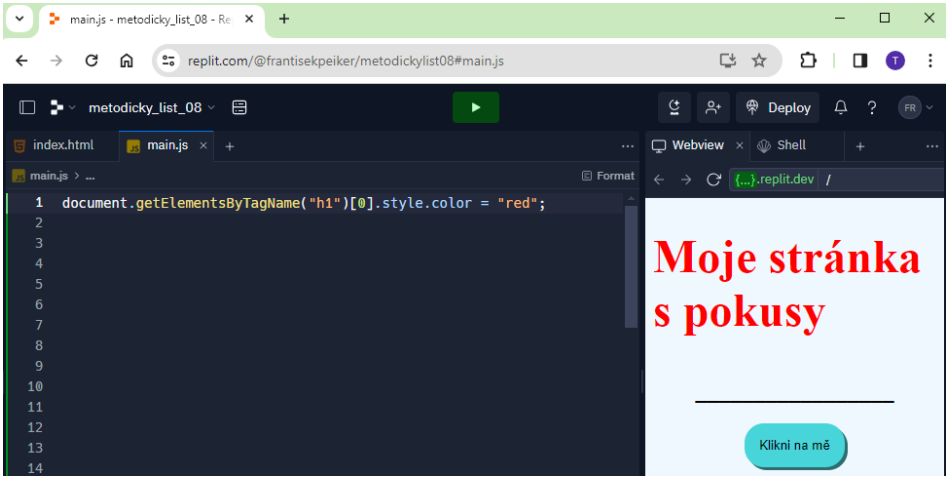
- Jak se anglicky řekne slovíčko „pokud“?
- K čemu by se mohly v programování využívat podmínky?
- Jak se odkazujeme na jednotlivé hodnoty v polích?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování odkazování se na HTML elementy pomocí JavaScriptu.
5. Představení podmínek žákům.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatné práci.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

Cílem této hodiny je seznámit žáky s podmínkami `if` a `else` v jazyce JavaScript. Nejprve je třeba zopakovat způsob odkazování se na HTML elementy pomocí JavaScriptových příkazů jako `document.getElementById()` a přidáním dalšího, kterým je `document.getElementsByTagName()`, který vyhledá všechny HTML elementy odpovídající požadavku a vrátí hodnotu v podobě pole, i tehdy, když najde pouze jeden

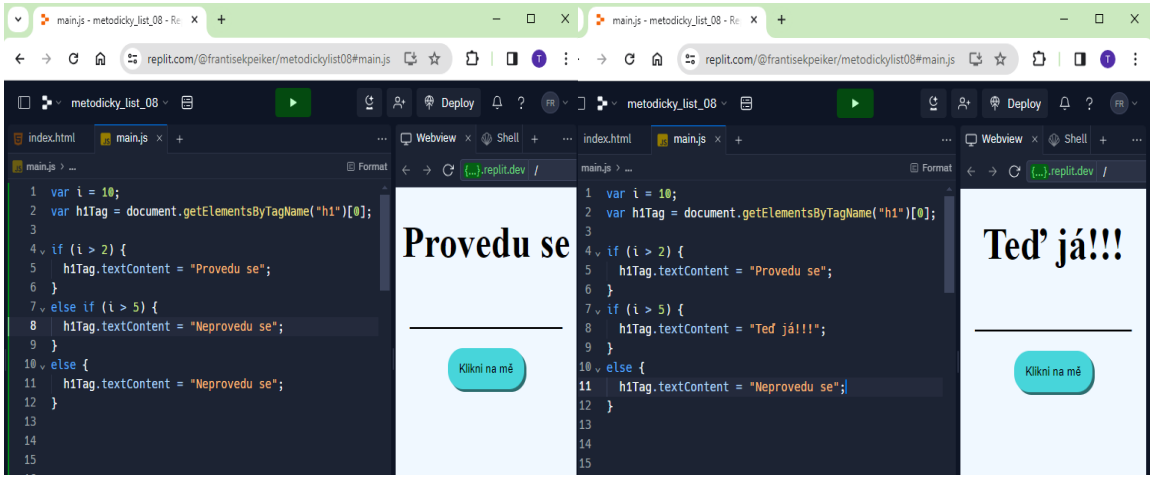
výsledek. K tomuto příkazu je tedy nutno navíc specifikovat, který z elementů z pole vybrat (viz Obrázek 21).



Obrázek 21: Příklad ukázky příkazu document.getElementsByTagName()

(Zdroj: vlastní zpracování)

Po dostatečném zopakování a procvičení nového příkazu učitel představuje žákům podmínky `if` a `else`. Vysvětluje na příkladech, kdy a proč je použít a jaký mají syntaktický zápis. Je možné napsat více podmínek `if` pod sebe, kdy JavaScriptový interpret použije podmínku, která se aplikuje jako poslední. Také existuje podmínka `else if`, používaná tehdy, kdy může nastat situace, ve které by první podmínka neovlivňovala ty další. Pokud je například potřeba zjistit, zda je určité číslo větší než 10, použila by se podmínka `if(cislo > 10)`. Kdyby po této podmínce následovalo `else if(cislo > 20)`, tak by se blok kódu pod touto podmínkou neprovedl, jelikož stále platí první podmínka, která říká, že číslo je větší než 10 (viz Obrázek 22).



Obrázek 22: Příklad funkce podmínek if, else if a else

(Zdroj: vlastní zpracování)

Po procvičení nových příkazů mohou žáci začít pracovat na samostatné práci, kterou je projekt umístěn na stránce GitHub na adrese https://github.com/Pikrpan/metodicky_list_08. Projekt lze stáhnout v případě používání lokálního vývojového prostředí nebo jej přímo importovat do vývojového prostředí Replit přes uvedený URL odkaz. V samostatném projektu žáci upravují pouze JavaScriptový soubor main.js podle zakomentovaných pokynů.

Žáky upravený soubor by mohl vypadat například následovně:

```
//Vytvoř proměnnou, která bude sloužit pro iteraci
var i = 0;

//Vytvoř proměnnou tak, aby se odkazovala na HTML element h2
var h2 = document.getElementsByTagName("h2")[0];
//Změň text elementu h2 tak, aby ukazoval, kolikrát si kliknul/a
na tlačítko
h2.textContent = "Klikl si na mě: " + i;

function prictiJedna(){
    //Zvyš hodnotu proměnné určené k iteraci o jedna
    i++;
    //Změň text elementu h2 tak, aby ukazoval, kolikrát si kliknul/a
na tlačítko
    h2.textContent = "Klikl si na mě: " + i;
    /*Napiš alespoň 3 podmínky, které změní barvu textu elementu h2
podle toho,
      kolikrát si na něj kliknul/a (10 kliknutí = modrá, 20 = červená,
...)*/
    if(i > 10){
        h2.style.color = "green";
    }
    if(i > 20){
        h2.style.color = "blue";
    }
    if(i > 30){
        h2.style.color = "purple";
```

```
    }
    if(i > 40){
        h2.style.color = "violet";
    }
    if(i > 50){
        h2.style.color = "red";
    }
    if(i > 60){
        h2.textContent = "Už stačí!!!"
        document.getElementById("btn").setAttribute("hidden", true);
    }
}
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.9 Metodický návrh č. 9 – Cykly

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript s využitím cyklů a příkazů pro práci s DOM, prostřednictvím kterého aktualizuje HTML elementy na webové stránce.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.
- Znalost proměnných a funkcí v jazyce JavaScript.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž, předvádění a pozorování.

Doporučené didaktické zásady:

- Soustavnosti, spojení teorie s praxí, trvalosti, uvědomělosti a aktivity.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut

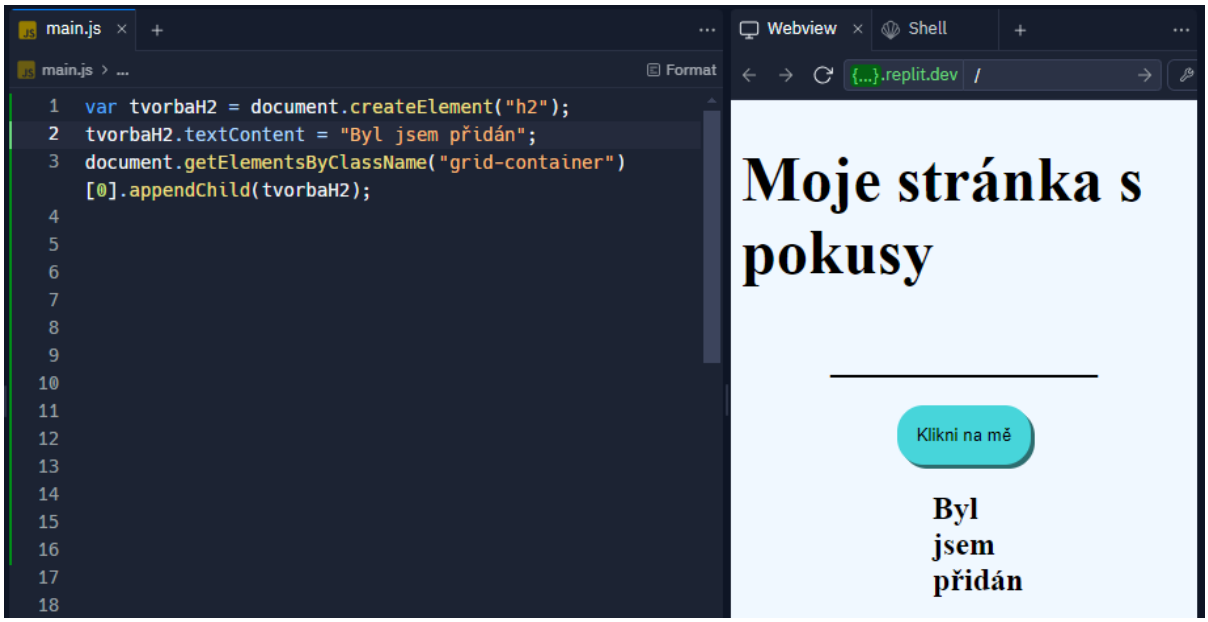
Úvodní otázky pro žáky:

- Jak se anglicky řekne „pro“ nebo „mezi tím“?
- K čemu asi v programování slouží cykly?

Průběh výuky:

1. Představení výukového cíle žákům.
2. Kladení úvodních otázek žákům.
3. Otevření vývojového prostředí.
4. Zopakování odkazování se na HTML elementy pomocí JavaScriptu.
5. Představení cyklů.
6. Žáci pracují podle pokynů učitele.
7. Žáci pracují na samostatné práci.
8. Prezentace výsledků a zhodnocení práce.
9. Zhodnocení samostatné práce žáky.

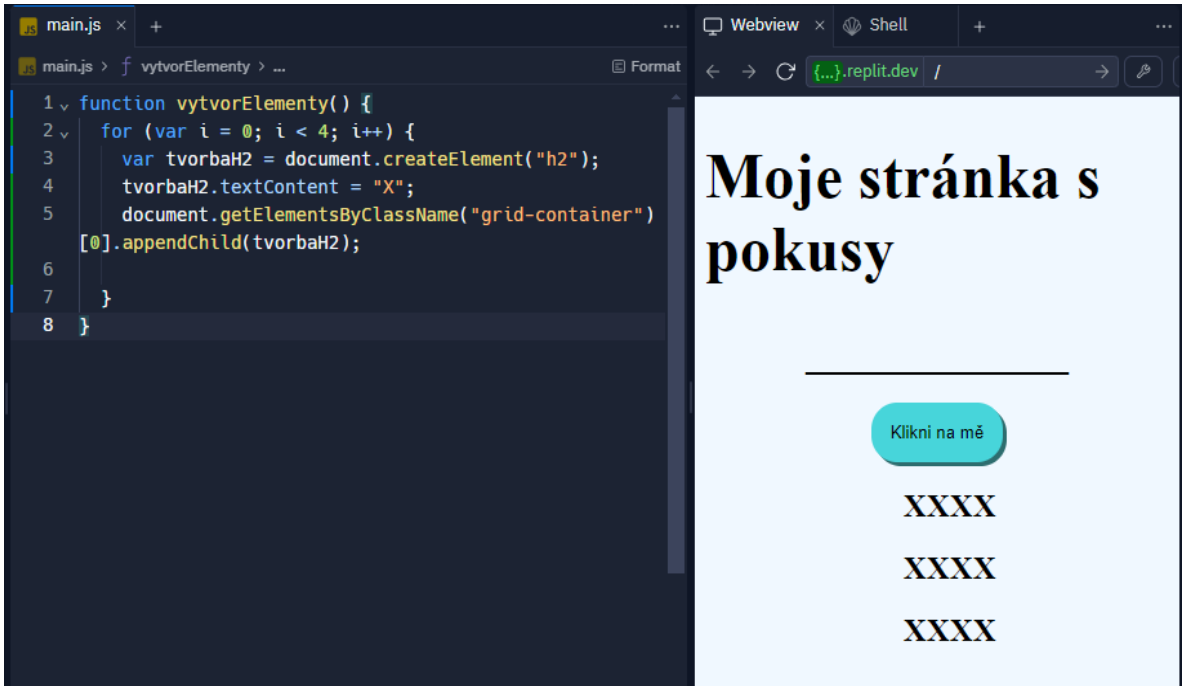
Cílem hodiny je seznámit žáky s JavaScriptovými cykly, především s cyklem `for`. Nejprve učitel s žáky zopakuje způsob odkazování se na HTML elementy pomocí JavaScriptových příkazů jako `document.getElementById()` nebo `document.getElementsByTagName()`. Učitel představí žákům nový příkaz `document.createElement()`, pomocí kterého se vytvářejí nové HTML elementy. Element vytvořen pomocí tohoto příkazu se dále musí vložit do DOM (Objektový Model Dokumentu), což lze provést dalším novým příkazem `.appendChild()` (viz Obrázek 23).



Obrázek 23: Příklad použití příkazů document.createElement() a .appendChild()

(Zdroj: vlastní zpracování)

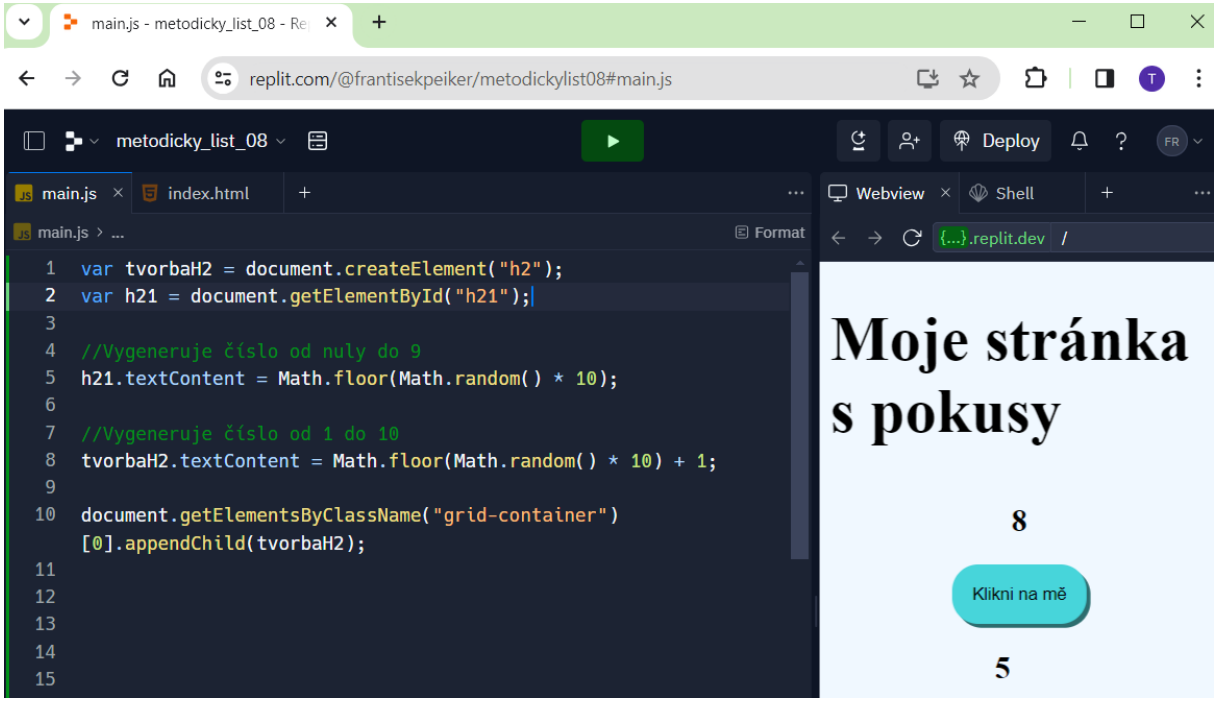
Po dostatečném zopakování a procvičení nových příkazů učitel představuje žákům cykly `for` a `while`. Vysvětluje na příkladech, kdy a proč který použít a jaký mají syntaktický zápis. Je důležité říct žákům o ob ou cyklech, ale většinou se v reálném světě setkají s cyklem `for`, proto je dobré se zaměřit především na něj a probrat ho do více do hloubky (viz Obrázek 24).



Obrázek 24: Příklad použití cyklu for uvnitř funkce

(Zdroj: vlastní zpracování)

Učitel dále představuje žákům další příkazy, kterými jsou `Math.floor()` a `Math.random()`. Příkaz `Math.floor()` zaokrouhluje dolů požadované číslo, `Math.random()` vygeneruje náhodné číslo mezi nulou a jedničkou. Po spojení těchto dvou příkazů lze vygenerovat náhodné číslo v zadaném rozmezí (viz. Obrázek 25).



Obrázek 25: Příklad použití příkazů Math.floor() a Math.random()

(Zdroj: vlastní zpracování)

Po procvičení nových příkazů mohou žáci začít pracovat na samostatné práci, kterou je projekt umístěn na stránce GitHub na adrese https://github.com/Pikrpan/metodicky_list_09. Projekt lze stáhnout v případě používání lokálního vývojového prostředí nebo jej přímo importovat do vývojového prostředí Replit přes uvedený URL odkaz. V samostatném projektu žáci upravují pouze JavaScriptový soubor main.js podle zakomentovaných pokynů.

Žáky upravený soubor by mohl vypadat například následovně:

```
//Vytvoř funkci s názvem vytvorElementy
function vytvorElementy(){
    //Vytvoř proměnnou, do které zapíšeš náhodnou hodnotu od jedné do pěti
    var nahodneCislo = Math.floor(Math.random() * 5 + 1)
    //Vytvoř cyklus for, který se má provést tolikrát, kolik je hodnota proměnné s náhodnou hodnotou
```

```
    for(var i = 0; i < nahodneCislo; i++){
        //Vytvoř HTML element h2
        var tvorbaH2 = document.createElement("h2");
        //Vytvořenému h2 elementu přiřaď hodnotu proměnné s náhodnou
hodnotou
        tvorbaH2.textContent = nahodneCislo;
        //Vlož h2 element do div elementu, který má třídu "grid-
container"
        document.getElementsByClassName("grid-
container")[0].appendChild(tvorbaH2);
    }
}
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

4.10 Metodický návrh č. 10 – Finální projekt – hod kostkou

Předmět:

- Informatika.

Výukový cíl:

- Žák vytvoří program v jazyce JavaScript, prostřednictvím přidá funkcionalitu webové stránce.

Tematický celek:

- Algoritmizace a programování.

Učivo:

- Algoritmizace, programování, tvorba digitálního obsahu.

Mezipředmětové vztahy:

- Matematika, anglický jazyk.

Vstupní znalosti žáků:

- Základní znalost blokově orientovaného programovacího jazyka.
- Základní znalost tvorby webových stránek.

Organizační forma:

- Hromadná/frontální výuka.

Výukové metody:

- Výklad, instruktáž.

Doporučené didaktické zásady:

- Soustavnosti, spojení teorie s praxí, trvalosti, uvědomělosti a aktivity.

Pomůcky:

- Počítače s připojením k internetu, projektor, pracovní list.

Časová dotace:

- 2 x 45 minut.

Průběh výuky:

1. Představení výukového cíle žákům.
2. Prezentace příkladu hotového projektu.
3. Otevření vývojového prostředí.
4. Žáci pracují na samostatné práci.
5. Prezentace výsledků a zhodnocení práce.
6. Zhodnocení samostatné práce žáky.

Cílem hodiny je projekt, na kterém žáci pracují samostatně nebo ve dvojicích, pokud si na něj netroufají sami. Učitel žákům předvede, jak by mohl hotový projekt vypadat (viz Obrázek 26). Na příklad hotového projektu lze nahlédnout na následujícím odkazu: https://frantaucinovouinformatiku.cz/my-work/kostky/index.html.

Obrázek 26: Ukázka hotového projektu

(Zdroj: vlastní zpracování)

Učitel společně s žáky otevřou vývojové prostředí, ve kterém si spustí připravené soubory samostatné práce. Soubory k projektu jsou umístěny na stránce GitHub na adrese https://github.com/Pikrpan/metodicky_list_10. Soubory lze stáhnout v případě používání lokálního vývojového prostředí nebo je přímo importovat do vývojového prostředí Replit pomocí uvedeného URL odkazu. Žákům je doporučeno upravovat pouze JavaScriptový soubor main.js podle pokynů v komentářích. Pokud jsou však žáci šikovnější, mohou si na stránku přidat různé HTML elementy, upravit pozadí nebo rozložení stránky, použít jiné obrázky apod.

Žáky upravený soubor by mohl vypadat například následovně:

```
//Přepiš text HTML elementu footer na své jméno
var footerText = document.getElementsByTagName("footer")[0];
footerText.textContent = "František Peiker"
```

```
//Vytvoř funkci s názvem zmenJmenaHracu
//Funkce zmenJmenaHracu změní text v odstavcích s id pPlayer1
a pPlayer2 na nově zadaný text
function zmenJmenaHracu() {
  var player1Name = prompt("Zadej jméno hráči 1: ");
  var player2Name = prompt("Zadej jméno hráči 2: ");

  document.getElementById("pPlayer1").textContent = player1Name;
  document.getElementById("pPlayer2").textContent = player2Name;
}

//Vytvoř funkci s názvem hodKostkami
/*
Tato funkce bude obsahovat minimálně:
  - dvě proměnné, které vygenerují náhodné číslo od jedné do šesti
(pro každého hráče zvlášť),
  - příkaz, který změní obrázek kostky podle hozeného čísla,
  - blok kódu, který porovná který hráč vyhrál a zapíše výsledek
do odstavce s id "pResult"

HODNĚ ŠTĚSTÍ!!
PS: Představivosti se meze nekladou.
*/
function hodKostkami() {
  var player1Number = Math.floor(Math.random() * 6 + 1);
  var player2Number = Math.floor(Math.random() * 6 + 1);

  var player1DiceVariable = document.getElementById("player1Dice");
  var player2DiceVariable = document.getElementById("player2Dice");

player1DiceVariable.setAttribute("src","./images/dice"+player1Number
+ ".png" );
```

```
player2DiceVariable.setAttribute("src","./images/dice"+player2Number
+ ".png");

  var pResultVariable = document.getElementById("pResult");

  if (player1Number > player2Number) {
    pResultVariable.textContent =
      document.getElementById("pPlayer1").textContent + " vyhrává!";
  } else if (player2Number > player1Number) {
    pResultVariable.textContent =
      document.getElementById("pPlayer2").textContent + " vyhrává!";
  } else {
    pResultVariable.textContent = "Nerozhodně!!!";
  }

}
```

Žáci prezentují učiteli svou práci a reflektují, zda pro ně cvičení bylo lehké nebo obtížné a jak jsou spokojeni se svou prací, zda by ji mohli vylepšit, popřípadě jak.

Poznámka:

- Pokud si žáci netroufají na samostatném cvičení pracovat sami, mohou vytvořit skupiny, ve kterých budou projekt zpracovávat. Ve skupinách si rozvrhnou, kdo bude tvořit jednotlivé části samostatné práce a vypracovávají je.

Závěr

Cílem diplomové práce bylo vytvořit ucelenou sadu metodických návrhů pro učitele informatiky na 2. stupni ZŠ se zaměřením na implementaci jazyka JavaScript do výuky v oblasti algoritmizace a programování.

Cíl diplomové práce byl splněn.

V teoretické části jsme se věnovali problematice algoritmizace a programování a její implementaci ve výuce na základních školách. Popsali jsme, jaké jsou možnosti výuky algoritmizace a programování a kdy je vhodné přejít z vizuálních blokově-orientovaných programovacích jazyků, jako Scratch, k plnohodnotným programovacím jazykům, jako Python, C# nebo JavaScript. V dalších kapitolách jsme podrobně popsali programovací jazyk JavaScript, jeho syntaxi, funkce, metody a další způsoby využití. Také jsme se zabývali jeho historií i aktuálními trendy v moderním vývoji webových aplikací. Dále jsme se věnovali metodice a didaktice, do které spadají výukové metody, organizační formy a didaktické zásady, které je vhodné dodržovat při výchovně-vzdělávacím procesu.

V praktické části jsme se zaměřili na tvorbu ucelené sady metodických návrhů zaměřených na jazyk JavaScript a jeho implementaci do výuky algoritmizace a programování na druhém stupni základních škol. Každý metodický návrh obsahuje stručné informace o daných příkazech a jsou v nich také uvedeny samostatné práce, které by měly žákům pomoci s upevňováním probraných příkazů. Metodické návrhy byly vytvořeny jako pomůcka k úvodu do jednotlivých problematik. Procvičování jednotlivých příkazů nemusí být v některých hodinách dostatečné, proto navíc doporučujeme jednotlivé metodické návrhy prostřídat s vlastními příklady. Díky tomu si žáci procvičí příkazy co možná nejvíce.

V prvních metodických návrzích jsme se zabývali především úvodem do jazyka JavaScript. Tyto metodické listy obsahovaly základní příkazy, jako například alert nebo prompt a základními koncepty, jako jsou proměnné a funkce. V dalších metodických návrzích se objevují jednotlivé příkazy pro odkazování se na jednotlivé HTML elementy. Příkazy jsou provázány z předchozími koncepty tak, aby si je žáci mohli procvičit společně s novými strukturami. Poslední metodický návrh obsahuje komplexní projekt, který musí žáci vyhotovit s využitím všech probraných konceptů z předchozích metodických návrhů. Finální projekt je koncipován jako hra, kterou mohou žáci využít i mimo školní prostředí. Žáci by měli být motivováni k jeho vytvoření a mohou si jej případně upravit podle svých schopností a představ.

Diplomová práce slouží jako výukový materiál pro implementaci programovacího jazyku JavaScript do výuky informatiky, ve které se výuka algoritmizace a programování vede moderními přístupy.

Seznam použitých zdrojů

Literatura

ADOBE, 2021. *Informace o webových aplikacích*. Online. Dostupné z: https://helpx.adobe.com/cz/dreamweaver/using/web-applications.html. [citováno 2024-01-28].

BROOKSHEAR, J. Glenn, David T. SMITH a Dennis BRYLOW, 2013. *Informatika*. Computer Press. ISBN 978-80-251-3805-2.

CLOKE, Harry, 1.6.2023. *Edgar Dale's Cone of Experience*. Online. Growth Engineering Technologies. Dostupné z: https://www.growthengineering.co.uk/what-is-edgar-dales-cone-of-experience/. [citováno 2024-04-07].

CODECADEMY, 2024. *What Is an IDE?*. Online. Dostupné z: https://www.codecademy.com/article/what-is-an-ide. [citováno 2024-04-07].

DENIS, Michael Aaron, 19.3.2024. *Tim Berners-Lee*. Online. Britannica. Dostupné z: https://www.britannica.com/biography/Tim-Berners-Lee. [citováno 2024-04-07].

DIGITÁLNÍ VZDĚLÁVÁNÍ, 15.11.2021. *Programování a informatické myšlení*. Online. YouTube. Dostupné z: https://www.youtube.com/watch?v=I9H-5FdRGK4&ab_channel=Digit%C3%A1ln%C3%ADvzd%C4%9Bl%C3%A1v%C3%A1n%C3%AD. [citováno 2024-04-07].

FLANAGAN, David, 2020. *JavaScript: the definitive guide: master the world's most-used programming language*. 7. vyd. O'Reilly. ISBN 978-1-491-95202-3.

GEEKSFORGEEKS, 2024. *DOM (Document Object Model)*. Online. Dostupné z: https://www.geeksforgeeks.org/dom-document-object-model/#methods-of-document-object. [citováno 2024-04-07].

GILLIS, Alexander 30.9.2018. *Integrated development environment (IDE)*. Online. TechTarget. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/integrated-development-environment. [citováno 2024-04-07].

KALHOUS, Zdeněk a Otto OBST, 2002. *Školní didaktika*. Portál. ISBN 807178253X.

KANTOR ILYA, 2024. *Interaction: alert, prompt, confirm*. Online. Dostupné z: https://javascript.info/alert-prompt-confirm. [citováno 2024-04-07].

KOLÁŘOVÁ, Julie, 2019. *Didaktické zásady ve výuce informatiky.* Online. DocPlayer.cz. Dostupné z: https://docplayer.cz/110450272-Didakticke-zasady-ve-vyuce-informatiky-didaktika-informatiky-1-prednaska-c-7-duben-2010.html. [citováno 2024-04-07].

LAURENČÍK, Marek, 2019. *Tvorba www stránek v HTML a CSS*. Grada. ISBN 9788027122417.

MAŇÁK, Josef a Vlastimil ŠVEC, 2003. *Výukové metody*. Paido. ISBN 80-7315-039-5.

MORITZ, Jeremy, 2018. *Code for Teens*. Mascot Books. ISBN 13: 9781684019601.

MOZILLA.ORG, 2024. *console: log() static method*. Online. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/console/log_static. [citováno 2024-04-07].

OLAWANLE, Joel, 1.9.2022. *What is an IDE? IDE Meaning in Coding*. Online. freeCodeCamp. Dostupné z: https://www.freecodecamp.org/news/what-is-an-ide-for-beginners/. [citováno 2024-04-07].

OTEVŘENÁ VĚDA, 23.8.2018. *Algoritmy – NEZkreslená věda IV.* Online. YouTube. Dostupné z: https://www.youtube.com/watch?v=HsO2reAF0IA&ab_channel=Otev%C5%99en%C3%A1v%C4%9Bda. [citováno 2024-04-07].

PEHLIVANIAN, Ara a Don NGUYEN, 2014. *JavaScript Okamžitě*. Computer Press. ISBN 978802514163.

PROGRAMMING WITH MOSH, 24.4.2018. *What is JavaScript?*. Online. YouTube. Dostupné z: https://www.youtube.com/watch?v=upDLs1sn7g4&t=61s&ab_channel=ProgrammingwithMosh. [citováno 2024-04-07].

ROBBINS, Jennifer Niederst, 2018. *Learning Web Design*. O'Reilly Media. ISBN 9781491960202.

SKALKOVÁ, Jarmila, 2007. *Obecná didaktika*. 2. vyd. Grada. ISBN 978-80-247-6981-3.

STACK EXCHANGE, 2024. *Technology: Programming, scripting, and markup languages*. Online. Dostupné z: https://survey.stackoverflow.co/2023/#technology-most-popular-technologies. [citováno 2024-04-07].

ŠTRÁFELDA JAN, 2024. *DOM*. Online. Dostupné z: https://www.strafelda.cz/dom. [citováno 2024-04-07].

UMÍME TO, 2024. *Algoritmické myšlení*. Online. Dostupné z: https://www.umimeinformatiku.cz/cviceni-algoritmicke-mysleni. [citováno 2024-04-07].

VORDERMAN, Carol, 2022. *Programování pro děti: od úplných základů k programování jednoduchých her*. Slovart. ISBN 978-80-276-0325-1.

WRÓBLEWSKI, Piotr, 2015. *Algoritmy*. Computer Press. ISBN 9788025141267.

W3SCHOOLS, 2024a. *JavaScript History*. Online. Dostupné z: https://www.w3schools.com/js/js_history.asp. [citováno 2024-01-28].

W3SCHOOLS, 2024b. *JavaScript Reserved Words*. Online. Dostupné z: https://www.w3schools.com/js/js_reserved.asp. [citováno 2024-04-07].

W3SCHOOLS, 2024c. *JavaScript String Reference*. Online. Dostupné z: https://www.w3schools.com/jsref/jsref_obj_string.asp. [citováno 2024-04-07].

ZORMANOVÁ, Lucie, 2012a. *Výukové metody v pedagogice: tradiční a inovativní metody, transmisivní a konstruktivistické pojetí výuky, klasifikace výukových metod*. Grada. ISBN 978-80-247-4100-0.

ZORMANOVÁ, Lucie, 1.2.2012b. *Výukové metody tradičního vyučování*. Online. Metodický portál RVP.CZ. Dostupné z: https://clanky.rvp.cz/clanek/c/s/15015/VYUKOVE-METODY-TRADICNIHO-VYUCOVANI.html. [citováno 2024-04-07].

ŽÁRA, Ondřej, 2015. *JavaScript Programátorské techniky a webové technologie*. Computer Press. ISBN 9788025145739.

Seznam obrázků

Seznam grafů

Seznam zkratek

ZŠ	základní škola
IFIP	mezinárodní federace pro zpracování informací
HTML	značkovací jazyk (Hypertext Markup Language)
CSS	kaskádové styly (Cascading Style Sheets)
ECMA	Evropská asociace výrobců počítačů
DOM	objektový model dokumentu
AJAX	asynchronní JavaScript a XML
XML	rozšiřitelný značkovací jazyk (Extensible Markup Language)
PHP	hypertextový preprocesor (Hypertext Preprocessor)
JSP	JavaServer Pages
SEO	optimalizace pro vyhledávače (Search Engine Optimization)
IDE	vývojové prostředí (Integrated Development Environment)
GUI	grafické uživatelské rozhraní (Graphical User Interface)
OS	operační systém (Operating System)

Anotace

Jméno a příjmení:	František Peiker
Katedra:	Katedra technické a informační výchovy
Vedoucí práce:	Mgr. Tomáš Dragon
Rok obhajoby:	2024

Název práce:	JavaScript jako nástroj pro výuku algoritmizace a programování na 2. stupni ZŠ
Název v angličtině:	JavaScript as a tool for teaching algorithmization and programming at lower secondary schools
Anotace práce:	Diplomová práce se zaměřuje na vytvoření ucelené sady metodických návrhů pro učitele informatiky na druhém stupni základních škol s důrazem na začlenění jazyka JavaScript do výuky algoritmizace a programování. V teoretické části práce je rozebrána problematika algoritmizace a programování na základních školách. Dále je zde detailně popsán jazyk JavaScript, jeho syntaxe, funkce, historie a jeho využití v moderním vývoji webových aplikací. V praktické části je prezentována ucelená sada metodických návrhů zaměřených na implementaci jazyka JavaScript do výuky algoritmizace a programování na druhém stupni základních škol. Každý metodický návrh obsahuje stručné informace o jednotlivých příkazech a samostatnou práci. V posledním metodickém návrhu se nachází komplexní projekt, ve kterém žáci upotřebí všechny předchozí koncepty.
Klíčová slova:	JavaScript, algoritmizace, programování, webové aplikace, výukové materiály
Anotace v angličtině:	The thesis focuses on the creation of a comprehensive set of methodological proposals for teachers of informatics at lower secondary schools with an emphasis on the inclusion of JavaScript in the teaching of algorithmization and programming. The theoretical part of the thesis deals with the issues of algorithmization and programming in primary schools. It also describes JavaScript in detail, its syntax, functions, history and its use in the development of modern web applications. The practical part presents a comprehensive set of methodological proposals aimed at implementing JavaScript in the teaching of algorithmization and programming at lower secondary. Each methodological proposal includes brief information about each concept and an individual work. The last methodological proposal

	contains complex project in the form of a game that incorporates all the previous concepts.
Klíčová slova v angličtině:	JavaScript, algorithmization, programming, web applications, teaching materials
Přílohy vázané v práci:	-
Rozsah práce:	84 stran
Jazyk práce:	čeština