

Univerzita Karlova
Pedagogická fakulta
Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Programovací jazyky ve výuce programování na SŠ
Programming languages in the teaching of programming in upper-secondary
schools

Michaela Meitnerová

Vedoucí práce: PhDr. Tomáš Jeřábek Ph. D.
Studijní program: Specializace v pedagogice (B7507)
Studijní obor: B IT (7507R040)

Odevzdáním této bakalářské práce na téma Programovací jazyky ve výuce programování na SŠ potvrzuji, že jsem ji vypracovala pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

Čelákovice 8. 7. 2024

Ráda bych poděkovala PhDr. Tomášovi Jeřábkovi Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Mé poděkování patří též přátelům a rodině.

ABSTRAKT

Tato bakalářská práce porovnává a uvádí příklady jednotlivých programovacích jazyků a nabízí možnosti jazyků, které jsou vhodné pro výuku programování na středních školách. V této práci se nachází nejen vysvětlení základních pojmů v programování, ale také historie programovacích jazyků, jejich rozdělení do programovacích paradigmat, podrobný popis každého jazyka a jeho funkcionality.

V poslední části bakalářské práce jsou uvedeny příklady programovacích jazyků a porovnání z hlediska syntaxe a využití v jednotlivých oborech IT. V práci navrhuji programovací jazyky, které je možné využít ve výuce konkrétních oblastech.

KLÍČOVÁ SLOVA

programování, programovací jazyky, výuka na středních školách

ABSTRACT

This bachelor thesis seeks to compare and give examples of different programming languages and to offer possible languages that are suitable for teaching programming in secondary schools. This thesis covers not only an explanation of the basic concepts in programming, but also a history of programming languages, their division into programming paradigms, as well as a detailed description of each language, including its functionality.

In the last part of the bachelor thesis, examples of programming languages and comparisons in terms of syntax and use in different IT domains are given. Lastly, programming languages that can be used in teaching specific domains are proposed.

KEYWORDS

programming, programming languages, teaching in secondary schools

Obsah

Úvod	8
1 Cíle, metody a struktura práce	9
2 Vymezení základní pojmů	10
3 Programovací jazyky	12
3.1 Struktura a význam programovacích jazyků	12
3.2 Historie a vývoj programovacích jazyků	12
3.3 Dělení programovacích jazyků	18
3.3.1 Vyšší a nižší programovací jazyky	18
3.3.2 Procedurální a neprocedurální programovací jazyky	20
3.3.3 Kompilované a interpretované	22
3.3.4 Strukturované programovací jazyky	24
3.3.5 Objektově-orientované programovací jazyky	25
3.3.6 Typová kontrola	27
3.4 Aplikační využití	28
3.4.1 Desktopové aplikace	28
3.4.2 Webové aplikace	29
3.4.3 Mobilní aplikace	29
3.4.4 Datová analýza a umělá inteligence	29
4 Výuka programovacích jazyků na SŠ	31
4.1 Rámcové vzdělávací programy a školní vzdělávací programy	31
4.2 RVP středního odborného vzdělávání	31
4.3 ŠVP středního odborného vzdělávání	32
5 Výběr programovacích jazyků	35
5.1 Jazyk C	36

5.1.1	Funkcionalita	37
5.1.2	Využití C	37
5.1.3	Syntaxe a sémantika	38
5.2	Jazyk C++	38
5.2.1	Funkcionalita	38
5.2.2	Využití C++	39
5.2.3	Syntaxe a sémantika	39
5.3	Jazyk C#	39
5.3.1	Funkcionalita	39
5.3.2	Využití C#	40
5.3.3	Syntaxe a sémantika	40
5.4	Java	40
5.4.1	Funkcionalita	41
5.4.2	Využití Javy	42
5.4.3	Syntaxe a sémantika	42
5.4.4	Groovy	43
5.4.5	Kotlin	43
5.5	JavaScript a TypeScript	43
5.5.1	Funkcionalita	44
5.5.2	Využití JavaScriptu a TypeScriptu	44
5.5.3	Syntaxe a sémantika	45
5.6	PHP	46
5.6.1	Funkcionalita	46
5.6.2	Využití PHP	46
5.6.3	Syntaxe a sémantika	47

5.7	Python.....	47
5.7.1	Funkcionalita	47
5.7.2	Využití Pythonu.....	48
5.7.3	Syntaxe a sémantika	48
6	Porovnání programovacích jazyků	49
6.1	Syntaxe programovacích jazyků.....	49
6.1.1	C	52
6.1.2	C++	56
6.1.3	C#	59
6.1.4	Java	62
6.1.5	JavaScript	65
6.1.6	PHP	68
6.1.7	Python.....	70
6.1.8	TypeScript	73
6.2	Možnosti využití	75
6.2.1	Desktopové aplikace.....	75
6.2.2	Webové aplikace.....	76
6.2.3	Mobilní aplikace	76
6.2.4	Datová analýza a umělá inteligence	77
	Závěr.....	79
	Seznam použitých informačních zdrojů	80
	Seznam obrázků.....	92

Úvod

Tato bakalářská práce porovnává programovací jazyky z hlediska vhodnosti jejich využití ve výuce programování na středních školách podle dvou základních faktorů – syntaxe a aplikačního využití. Tato práce je striktně zaměřena na textové programovací jazyky.

V roce 2023 proběhla velká změna RVP SOV v oblasti ICT, jehož nedílnou součástí je právě výuka programování. Zavádí se zde nové pojetí Informatického vzdělávání a zvyšuje se minimální počet vyučovacích hodin. V RVP je programování pojato velmi obecně a není zde specifikován programovací jazyk, který by měl být vyučován. Proto je vhodné nabídnout i konkrétní doporučení pro učitele při výběru jazyka.

Programovacích jazyků je velké množství a každý nabízí jiný programovací styl a možnosti aplikačního využití. Pro výběr správného programovacího jazyka je důležité znát jeho programovací paradigma, syntaxi a jeho využití v daném oboru, který studenti mají vybraný.

Inspirací pro tuto bakalářskou práci byl podnět od autorčiných kolegů, zdali existuje jeden univerzální programovací jazyk, který by se dal učit na všech středních školách a zároveň splňoval veškeré podmínky. Vzhledem k tomu, že je výběr vždy čistě na daném učiteli, může tato práce sloužit jako přehled možností, kde si každý učitel může vybrat dle svého vlastního úsudku a potřebám.

1 Cíle, metody a struktura práce

Hlavním cílem práce je přispět ke zlepšení výuky programování na SŠ skrze nalezení vhodných programovacích jazyků, které je možné pro programování na středních školách využít.

Díložními cíli jsou: najít vhodné jazyky pro výuku na SŠ a porovnat jednotlivé jazyky z různých hledisek.

Tato práce se zaměřuje na podrobný popis programovacích jazyků. V první části se zabývá jejich dělením dle různých aspektů a představením jednotlivých jazyků. V druhé části se rozebírá jejich využití v reálném světě a příklady jejich syntaxí. Výsledkem práce by mělo být zhodnocení programovacích jazyků z hlediska jednoduchosti pro začínajícího programátora.

Práce definuje základní pojmy nezbytné pro programování, rozděluje programovací jazyky dle různých paradigmat, zdůrazňuje jejich rozdíly a historii, a popisuje jejich využití. Tento přístup poskytuje základní vřled do problematiky programovacích jazyků.

Další část bakalářské práce obsahuje zmapování RVP a ŠVP, kde se podrobněji rozebírá výuka a následně výběr programovacích jazyků, které by mohly být vhodné pro začátek výuky programování. Každý jazyk je zde specifikován z hlediska jeho syntaxe a funkcionality.

Poslední kapitola se zabývá samotným porovnáním programovacích jazyků. Kapitola obsahuje základní zdrojový kód, který je napsán vždy podle syntaxe daného programovacího jazyku. Jednotlivé příkazy byly vybrány na základě potřeb RVP. Díky tomuto lze jasně vidět přehlednost a složitost psaní kódu. Nakonec jsou jazyky porovnány z hlediska jejich využití v oblasti aplikačního vývoje IT.

2 Vymezení základní pojmů

Pro orientaci v dané problematice je zapotřebí vymezit základní pojmy používané při programování. Během výuky programování se studenti setkávají s velkým množstvím pojmů, které je zapotřebí vysvětlit a dát do kontextu pro jejich lepší pochopení. V této kapitole jsou vysvětleny nejzákladnější pojmy v rámci informatiky.

Programovací jazyk je soubor instrukcí napsaných programátorem, který dává počítači pokyny k provedení a splnění určitého úkolu. Tento soubor instrukcí je kód strukturovaný podle určité syntaxe programovacího jazyka. (Tuama, 2023)

Programovací jazyky pro počítače, které umožňují vytvářet skripty – softwarové instrukce – pro použití v aplikacích a na webových stránkách, se nazývají skriptovací jazyky. Skriptovací engine, program nebo aplikace zpracovává příkazy skriptu jeden po druhém v běhovém prostředí. Tyto instrukce jsou překládány do čitelného strojového kódu, který může procesor za běhu vykonávat pomocí interpretu. (Bharadwaj, 2023)

Programování je proces vytváření souboru instrukcí, které počítači říkají, jak má provést určitý úkol. Během programování probíhá analýza, generování algoritmů, profilování přesnosti algoritmů a jejich samotná implementace. Obvykle ve zvoleném programovacím jazyce. (Simplilearn, 2023)

Při programování se lze setkat s pojmem algoritmus, se kterým pak programátor pracuje nebo ho sám vytváří. Algoritmem je možno nazvat postup, který se používá k řešení nějakého problému, k provádění výpočtu nebo k dosažení určitého cíle. Algoritmy fungují jako přesně daný seznam instrukcí, které se mají provést pro určité činnosti krok za krokem.

„Algoritmus je nejen konečnou množinou pravidel, která popisují posloupnost operací pro řešení jistého typu problémů, ale zároveň musí splňovat pět důležitých vlastností: konečnost, určitost, vstup, výstup a efektivitu.“ (Knuth, 2008)

Konečným výsledkem programování je program, který už může být spustitelný pro běžného uživatele. Program je pak samotný soubor instrukcí v programovacím jazyce, které má počítač provést. Počítačové programy jsou jednou ze součástí softwaru, který zahrnuje také dokumentaci a další nehmotné součásti. Programátor vytváří zdrojový kód daného

programu. Ten potřebuje ke svému spuštění jiný počítačový program, protože počítače mohou spouštět pouze své nativní strojové instrukce. (Wilkins, 2021)

„Počítačový program vznikne tak, že algoritmus vyjádřený v programovacím jazyce převedeme do tvaru spustitelného na počítači. Přejít od algoritmu zapsaného v programovacím jazyce k programu spustitelnému na počítači se nazývá překlad a provádí jej speciální program označovaný jako překladač (kompilátor).“ (Matoušek, 2016)

Vývoj počítačových programů probíhá většinou ve specializovaných vývojových prostředích, známých jako integrovaná vývojová prostředí (Integrated Development Environment)¹. Tato prostředí poskytují nástroje, které usnadňují psaní, testování a ladění kódu. Přesto lze programovat i v jednodušších editorech, jako je obyčejný poznámkový blok, což ukazuje na univerzálnost a dostupnost programování jako dovednosti. IDE však poskytují výhody, jako je automatické doplňování kódu, nástroje pro správu verzí a integrované ladící programy, což výrazně zvyšuje produktivitu programátorů a přináší mnoho dalších výhod.

„IDE znamená pro programátora komfort, kdy v rámci jediného programu (přesněji aplikace) zapíše zdrojový text, nastaví (pokud je to nutné) parametry překladu, provede překlad, testuje a ladí činnost hotového programu.“ (Matoušek, 2016)

¹ IDE – vývojové prostředí

3 Programovací jazyky

Existuje celá řada programovacích jazyků, ze kterých je možno vybírat. Každý programovací jazyk nabízí jiné možnosti využití, má jinou syntaxi a nabízí jinou přívětivost během programování. Je zapotřebí se seznámit se strukturou programovacích jazyků, jejich historií a dělením pro jednodušší porovnání. Díky tomu je pak snazší pochopit jejich účel a funkčnost, která má vliv právě i při výuce. V této kapitole jsou jednotlivě popsány a specifikovány kategorie, do kterých se jazyky dělí a vysvětleny rozdíly mezi nimi.

3.1 Struktura a význam programovacích jazyků

Při programování je důležité dodržovat pravidla, která určují, jakým způsobem lze psát funkční a efektivní kód. Základem pro správný zápis programu v jakémkoli programovacím jazyce jsou dvě klíčové složky: syntaxe a sémantika. Porozumění syntaxi a sémantice je zásadní pro psaní správně fungujícího kódu.

Syntaxe představuje formální pravidla, která určují, jak správně kombinovat symboly, proměnné a další jazykové prvky do správně strukturovaného kódu. Syntaxe určuje strukturu kódu podobně jako gramatická pravidla v přirozeném jazyce. (Rouse, 2018a)

Oproti tomu sémantika dává kombinovaným řetězcům význam. Sémantika je termín pro způsoby prezentace dat a příkazů. Sémantika je lingvistický pojem oddělený od pojmu syntaxe, který je také často spojován s atributy počítačových programovacích jazyků. Pokud je použit v kódu pojem, který počítači nedává smysl, jedná se o „sémantickou chybu“. Zjednodušeně lze říci, že sémantika je logický význam příkazů. (Rouse, 2018b)

3.2 Historie a vývoj programovacích jazyků

První programovací jazyk vznikl již v 19. století. Ačkoliv ho nelze klasifikovat jako klasický programovací jazyk, který je znám dnes, lze říci, že fungoval na podobném principu jako dnešní programovací jazyky. Vznikl pro Analytický stroj Charlese Babbage v roce 1843 a měl sloužit k vypočítání Bernoulliho čísel. (Hierso, 2022)

Navrhla ho Ada Lovelace, britská aristokratka, která byla vzdělaná nejen v matematice ale i ve vědě. Jednalo se tenkrát o překlad psaného díla o Analytickém stroji. Ada dílo přeložila a dopsala k němu své vlastní poznámky, které samotné dílo třikrát přesahovalo. Popsala

v nich, jak by měl stroj využívat děrované štítky pro výpočty, které by měly obsahovat instrukce, paměťové jednotky a další komponenty. K tomu také dopsala algoritmy, a právě tím vznikl první programovací jazyk. Kromě toho vynalezla větvení programu a využití podprogramů. (Králová, 2007)

Jejím kolegou byl vynikající matematik, vynálezce a strojní inženýr Charles Babbage, který v roce 1837 navrhl analytický stroj, jenž byl návrhem univerzálního, programovatelného, parou poháněného výpočetního stroje. Ten obsahoval aritmetickou logiku, podmíněné větvení, smyčky, integrovanou paměť a která by ke vstupu a výstupu dat používal děrné štítky. (Ryan, 2019)

Práci Ada Lovelace byl ovlivněn Alan Turing další britský matematik, který je dnes považován za otce informatiky a umělé inteligence. Vynalezl Turingův stroj, který lze považovat za model univerzálního počítače. Dále je velmi oceňován za jeho práci pro britské středisko pro rozluštění kódů, kde přispěl k zdokonalení technik luštění šifer slavného stroje Enigma. (Deneire, 2021)

Turingův stroj je abstraktní koncept jednoduchého výpočetního zařízení, které má zkoumat rozsah o omezení toho, co lze vypočítat. Jeho automatické stroje, které popsal v roce 1936, měly pomoci s výpočty reálných čísel. Dnes jsou považovány za jedny ze základních modelů teoretické informatiky². (Daylight, 2015)

Myšlenka, že jakýkoliv stroj pro obecné použití lze v zásadě modelovat jako univerzální Turingův stroj, se stala důležitým principem v kontextu automatického programování v 50. letech 20. století. V kontextu programování pak šlo o myšlenku, že lze sestavit stroj, který je schopen „napodobit“ chování jakéhokoliv jiného stroje, a tedy v konečném důsledku o zaměnitelnost mezi hardwarem stroje a implementací jazyka. (Daylight, 2015)

Oproti teoretickému přístupu Alana Turinga se Konrad Zuse věnoval sestrojování počítačů, byl známý především díky jeho prvním funkčním programovatelným počítačím strojům. Všechny jeho počítače využívaly binární reprezentaci, díky čemuž bylo zpracování

² Teoretická informatika – obor matematické informatiky, který se zaměřuje na abstraktnější a matematické aspekty zpracování informací

základních logických operátorů (AND, OR, NOT) velmi snadné. (History Computer Staff, 2023)

Kromě sestrojování počítačů Zuse vymyslel první reálný programovací jazyk, který se jmenuje Plankalkül v začátcích 40. let minulého století. Plankalkül je považován za první vysokoúrovňový programovací jazyk, který se bohužel nesetkal s velkým komerčním úspěchem, částečně kvůli tomu, že sám autor Zuse se věnoval vyvinutí nového počítače Z3, a proto nikdy nebyl plně využitý jeho pravý potenciál a zůstal pouze na papíře. (History Computer Staff, 2023)

Prvním široce využívaným programovacím jazykem se stal v roce 1949 nízkoúrovňový jazyk Assembler neboli jazyk symbolických adres, který měl usnadnit pochopení strojového kódu neboli přesných instrukcí potřebných k fungování počítače. (Kidder, 2023)

Programovací jazyky 50. let 20. století

Rané počítačové jazyky se vyskytovaly v mnoha variantách, z nichž všechny byly zahrnuty pod obecný termín: autokód. První kompilovaný programovací jazyk, autocode, debutoval v roce 1952 a mohl být převeden přímo do strojového kódu pomocí výše uvedeného nástroje respektive kompilátoru. (Kidder, 2023)

V roce 1957 vyvinul John Backus program FORMula TRANslation neboli FORTRAN. Tento programovací jazyk z padesátých let se stále používá v superpočítačích a při matematických a vědeckých výpočtech. (Roller, 2022)

Američtí a evropští vědci vyvinuli v roce 1958 algoritmický jazyk známý pod názvem ALGOL, který později posloužil jako základ známých programovacích jazyků, například Pascal, Java, C a C++. ALGOL nově představil blokovou strukturu, která byla součástí jazyka. Program se skládá z jednotlivých bloků, které mají stejnou strukturu jako celý program a mohou obsahovat data a instrukce. (Hemmendinger, 2023a) John McCarthy vytvořil ve stejném roce procesor List neboli LISP. Firmy Boeing a Genworks jej stále používají, přestože byl určen pro umělou inteligenci. (Roller, 2022)

Dalším jazykem COBOL, což je zkratka pro Common Business Oriented Language a byl vytvořen v roce 1959 skupinou pod vedením Dr. Grace Murray Hopperové. Byl používán pro telefonní hovory, dopravní signály a zpracovatele transakcí kreditními kartami.

Programovací jazyky jako je COBOL, jsou základem dnešních virtuálních ústředí a dalších špičkových komunikačních řešení. (Roller, 2022)

Skupina studentů Dartmouth College vytvořila program Beginners All-Purpose Symbolic Instruction Code neboli BASIC. Jazyk byl určen pro studenty, kteří neměli silné základy matematiky nebo informatiky. Tento jazyk dále rozvíjeli zakladatelé společnosti Microsoft Bill Gates a Paul Allen a díky tomu se stal prvním produktem této společnosti, který se dostal na trh. (McCracken, 2014)

Programovací jazyky 70. let 20. století

Niklaus Wirth vytvořil v 70. letech 20. století počítačový programovací jazyk Pascal pro výuku strukturovaného programování. Jazyk klade důraz na systematické používání řídicích struktur nebo smyček a podmíněných příkazů bez nutnosti příkazů GOTO. Notace jazyka Pascal byla podobná notaci jazyka ALGOL, ale lišila se tím od jazyků ALGOL, FORTRAN a COBOL, že umožňovala definovat datové typy. Programátoři mohli zavést názvy pro složitá data prostřednictvím uživatelsky definovaných datových typů, jejichž správné použití pak mohl překladač jazyka ověřit před spuštěním programu. (Hemmendinger, 2023b)

Rok 1972 můžeme považovat za velmi důležitý pro vývoj programovacích jazyků. Byly vytvořeny hned tři: SmallTalk, SQL a jazyk C. Smalltalk vytvořili ve výzkumném centru Xerox Palo Alto v Alan Kay, Adele Goldberg a Dan Ingalls. Jeho účelem bylo upravovat kód. Jeho principy lze nalézt v moderních jazycích Java, Python a Ruby a používají ho společnosti jako CrowdStrike a Logitech. Jazyk C byl vytvořen Dennisem Ritchie v Bell Telephone Laboratories. Byl vytvořen pro práci s operačním systémem Unix. Raymond Boyce a Donald Chamberlain jazyk SQL (Structured Query Language). V době svého vzniku byl znám pod názvem SEQUEL a sloužil k prohlížení a úpravě dat v databázích. (Roller, 2022)

Tyto tři jazyky znamenají významný pokrok, který ovlivnil některé z největších technologických společností na světě v současnosti. Tyto pokroky ovlivnily všechny aspekty našeho digitálního života od sociálních médií až po programy pro doporučování vedoucích pracovníků ve firmách. (Roller, 2022)

Programovací jazyky 80. let 20. století

V 80. letech vznikl programovací jazyk ADA, pojmenovaný po průkopnici výpočetní techniky Adě Lovelace. Vytvořila ho skupina Jeana Ichbiaha na CUU Honeywell Bull. (Roller, 2022) ADA je strukturovaný, staticky typovaný, imperativní a objektově orientovaný programovací jazyk. Používá se v mnoha vesmírných a dopravních projektech, včetně systémů řízení letového provozu v Austrálii, Belgii a Německu. (Lestál, 2023)

V roce 1985 byla vydána první verze jazyka C++, který je rozšíření jazyka C, které zahrnuje možnosti jako tvorbu tříd, virtuální funkce a šablony. Bjarne Stroustrup v Bellových laboratořích provedl úpravy jazyka C. Od roku 1986 se trvale umísťuje mezi deseti nejlepšími programovacími jazyky a v roce 2003 byl uveden do síně slávy. Jazyk C++ je použit jako základ pro spoustu vysoce výkonného softwaru, příkladem může být Adobe Photoshop, Unreal Engine a podobně. (Lestál, 2023)

O dva roky později vytvořil Larry Wall PERL jako vysokoúrovňový jazyk pro obecné použití, lze v něm vytvářet prakticky cokoli. Ačkoli jej zamýšlel pro editaci textu, nyní se používá především pro databázové aplikace, zpracování dat, grafické programování a správu systémů. (Roller, 2022)

Programovací jazyky 90. let 20. století

Haskell je funkcionální programovací jazyk pojmenovaný po Haskellu Brooksu Currym, americkém logikovi a matematikovi. Jedná se o čistě funkcionální programovací jazyk, což znamená, že je především matematické povahy. Je široce používán v různých odvětvích, zejména v těch, které se zabývají složitými výpočty a záznamy. Haskell, stejně jako mnoho dalších programovacích jazyků z této doby, se běžně nepoužívá pro známé aplikace. Ačkoliv tento programovací jazyk nebyl vyvinut pro programování her, byla v něm například vyvinuta hra Nikki and the Robots. (Roller, 2022)

Python byl vytvořen v roce 1991 Guidem Van Rossumem a je pojmenován po britské komediální skupině Monty Python. Jedná se o univerzální vysokoúrovňový programovací jazyk navržený tak, aby podporoval širokou škálu programovacích stylů a zároveň se příjemně používal. (Lestál, 2023)

Ve stejném roce byl vytvořen programovací jazyk Visual Basic, který umožňuje programátorům používat styl drag-and-drop³. Díky tomu lze vybírat a měnit předem připravené části kódu prostřednictvím grafického uživatelského rozhraní (GUI). Tento jazyk se v dnešní době příliš nepoužívá, nicméně společnost Microsoft využila část jazyka Visual Basic do řady svých aplikací, jako jsou Word, Excel a Access.

O dva roky později byl vyvinutý Ruby, interpretovaný vysokoúrovňový programovací jazyk, vytvořený Yukihirom Matsumotem. Jedná se o výukový jazyk, který byl mimo jiné inspirován jazyky Ada, Lisp, Perl a Smalltalk. Byl navržen pro editaci textu, ale dnes se používá především v databázových aplikacích, při zpracování dat, grafickém programování a správě systémů. (Lestál, 2023)

V roce 1995 byly vyvinuté další programovací jazyky, který se dnes běžně využívají a to Java, PHP a JavaScript. (Roller, 2022)

Programovací jazyky 21. století

Jazyk C# byl poprvé vydán v roce 2000 jako součást Microsoft frameworku .NET a od té doby se stal velmi oblíbeným jazykem. Používá se téměř ve všech produktech společnosti Microsoft a používá se především při vývoji desktopových aplikací. (Lestál, 2023)

Scala je programovací jazyk vytvořený v roce 2003 Martinem Oderským, který kombinuje strukturované objektově orientované programování s matematickým funkcionálním programováním. Protože je Scala kompatibilní s Javou, je užitečná při vývoji aplikací pro Android. (Lestál, 2023)

Jazyk Go je imperativní programovací jazyk. Vytvořili ho v roce 2007 Robert Griesemer, Rob Pike a Ken Thompson ve společnosti Google, ale jako open-source programovací jazyk byl uvolněn až v roce 2009. Je navržen tak, aby byl jednoduchý, efektivní a snadno se učil, což z něj činí oblíbenou volbu pro vytváření škálovatelných síťových služeb, webových aplikací a nástrojů příkazového řádku. (Yadav, 2023a)

Jedním z nejmladších programovacích jazyků je Swift. Apple vytvořil Swift v roce 2014, aby nahradil jazyky C, C++ a Objective-C. Jeho cílem bylo, aby byl jednodušší a méně

³ Drag-and-drop – anglický výraz pro přetáhnout a pustit

náchylný k chybám než předchozí jazyky. Díky své přizpůsobivosti lze využít pro cloudové, mobilní i desktopové aplikace. (Lestál, 2023)

3.3 Dělení programovacích jazyků

Programovací jazyky lze dělit podle spousty kritérií. Je důležité zmínit, že některé programovací jazyky umožňují kombinovat různé přístupy a nelze je zcela zařadit do jedné skupiny.

Programovací jazyky je možno dělit podle míry abstrakce (vyšší a nižší programovací jazyky), způsobu překladu a spuštění (kompilované a interpretované programovací jazyky) a oblasti použití (univerzální jazyky a doménově specifické). (Coursera, 2023a)

Vyšší programovací jazyky pak lze dělit na procedurální (imperativní) a neprocedurální (deklarativní). Procedurální jazyky se rozlišují dále na strukturované a objektově orientované. Neprocedurální jazyky se rozdělují na funkcionální a logické. (Coursera, 2023a)

Některé programovací jazyky byly vytvořeny čistě pro výuku programování a pro zlepšení algoritmického myšlení (například Baltík, Karel, Scratch nebo Kodu Game Lab). Jiné programovací jazyky slouží pro zábavu v programátorské komunitě. Takové jazyky se nazývají esoterické (například Cheese++, WhiteSpace, Feta nebo Danica).

3.3.1 Vyšší a nižší programovací jazyky

Programovací jazyky je možno dělit na základě blízkosti k systémovému hardwaru (především procesoru) na vyšší a nižší. Většina programovacích jazyků, které jsou dnes nejznámější, se řadí mezi vyšší programovací jazyky. Je to především kvůli tomu, že vyšší programovací jazyky jsou uzpůsobeny tomu, aby bylo pro programátora jednoduché převést algoritmus do programového kódu. Nižší programovací jazyky jsou strojově orientovány. To znamená, že je potřeba přesně popsat proces k provedení konkrétní úlohy. Algoritmus musí být specifikován z hlediska možností a specifikací procesoru. (Isaac Computer Science, 2019)

Vyšší programovací jazyky

Vyšší programovací jazyky jsou určeny k zjednodušení programování počítačů. Lze říci, že jsou navrženy pro jednoduchou komunikaci člověka s počítačem. Zdrojové kódy se skládají ze snadno čitelných struktur, které jsou později přeloženy do nižšího programovacího jazyka. Takový kód může dále spouštět pouze konkrétní procesor. Tyto jazyky umožňují vytvářet program v mnohem srozumitelnějším zápisu pro člověka a je zpravidla nezávislý na hardwarové architektuře. Ovšem každý řádek kódu, který je napsán ve vyšším programovacím jazyce, musí být před implementací počítačem přeložen do strojového jazyka. (Priya, 2023)

Vyššími programovacími jazyky jsou například C#, Java, JavaScript, Pascal, PHP, Python, Swift a mnoho dalších. Každý z těchto programovacích jazyků ale používá odlišný zápis a využívají se pro jiný vývoj aplikací. Některé jsou vhodnější pro tvorbu webových stránek, jiné jsou určeny pro tvorbu programů pro desktopové nebo mobilní aplikace. Všechny ale mají společné to, že před implementací kódu musí být zpracovány kompilátorem nebo interpretem. (Priya, 2023)

Nižší programovací jazyky

Nižší programovací jazyky využívají specifickou instrukční sadu procesoru. Tyto instrukce jsou jiné pro každý procesor a jsou definovány výrobcem. Což znamená, že kód, který je napsán v nižším programovacím jazyce lze spustit pouze na procesoru stejného typu, pro který byl napsán. Kódy jsou proto nepřenositelné a na počítači s jiným procesorem nejsou spustitelné. Jsou ale nejbližší k přirozenému neboli binárnímu jazyku počítače. Tyto jazyky se dělí na assembler (jazyk symbolických instrukcí) a strojový jazyk. (Panigrahi, 2023)

Jednou z výhod těchto programovacích jazyků je rychlost programu nebo možnost pracovat přímo s paměťovými místy. Nevýhodou je jejich náročnost na programování a často ne příliš srozumitelný zápis pro člověka. (Panigrahi, 2023)

V současné době se tyto jazyky využívají spíše výjimečně. Lze se s nimi setkat při psaní jader operačních systémů nebo efektivních programů pro řídicí jednočipové mikropočítače, nebo pro psaní ovladačů zařízení. (Panigrahi, 2023)

Strojový kód

Strojový jazyk, známý také jako objektový kód nebo strojový kód, je soubor bitů, které počítač dokáže číst a interpretovat. Mnoho běžně používaných programovacích jazyků je nakonec přeloženo do strojového jazyka, aby jim počítače rozuměly. Počítače však tyto jazyky nedokážou přímo interpretovat, a proto je nutné je nejprve zkompileovat. Kompilací se rozumí překlad do strojového jazyka, kterému počítač rozumí a může jej vykonat. (Ashtari, 2023)

Assembler

Assembler je jazyk nejnižší úrovně, tzv. „jazyk symbolických adres“. Jedná se o programovací jazyk, který umožňuje psát programy přímo ve strojovém kódu procesoru. Program napsaný v assembleru se skládá ze strojových instrukcí. Jedna instrukce je určitým povelům pro procesor, aby vykonal určitou činnost. Programy, které jsou napsané v assembleru není možné přenést na jiný procesor, než pro který byl napsán, pokud oba procesory nejsou vyrobeny stejnou architekturou. (Zezula, 2003)

Rozdíl mezi strojovým kódem a assemblerem

Zatímco strojový jazyk je binární jazyk, kterému počítače rozumí, jazyk assembleru se před spuštěním překládá. Jazyky assembleru předběhly o krok strojový jazyk. Odstranily většinu zdoluhavých, časově náročných a k chybám náchylných operací, které se vyskytovaly při programování první generace počítačů. Zbavily programátory úkolů, jako je výpočet adres a pamatování si číselných kódů, a staly se tak standardem pro mnoho typů programování. (Ashtari, 2023)

Jedním z klíčových rozdílů mezi oběma jazyky je úroveň abstrakce. Strojový jazyk je mnohem blíže hardwaru, skládá se z řady binárních instrukcí, které může procesor přímo provádět. Na druhou stranu jazyk assembleru používá k reprezentaci instrukcí strojového jazyka mnemotechnické znaky a symboly, což usnadňuje čtení a zápis. (Ashtari, 2023)

3.3.2 Procedurální a neprocedurální programovací jazyky

Procedurální a neprocedurální jazyky se hojně používají při vývoji aplikací a dalšího počítačového softwaru. (Panigrahi, 2022a)

Tyto jazyky jsou dnes výpočetními modely pro specifikaci většiny programování. Hlavní rozdíl mezi těmito výpočetními modely spočívá v tom, že procedurální jazyk je orientován na příkazy, zatímco neprocedurální jazyk je orientován na funkce. Procedurální programovací jazyky navíc provádějí výpočet jako posloupnost příkazů, které manipulují s uloženými daty, dokud není dosaženo požadovaného výsledku. Naproti tomu neprocedurální jazyk představuje programy jako vztahy mezi matematickými výrazy, které jsou založeny na závislostech. (Neha, 2023)

Procedurální programovací jazyky

Procedurální jazyky, ve kterých jsou programy předem definovanou sadou instrukcí. Tyto instrukce jsou prováděny počítači ve stejném pořadí, v jakém byly zapsány. Každá instrukce říká počítači, co přesně a jakým způsobem má vykonat. (Basharat, 2022)

Procedurální jazyk rozděluje program na proměnné, funkce, příkazy a podmíněné operátory. Procedury nebo funkce jsou implementovány na datech a proměnných za účelem provedení úlohy. Tyto procedury mohou být volány kdekoli v hierarchii programu a také jinými procedurami. Program napsaný v procedurálním jazyce obsahuje jednu nebo více procedur. (Rouse, 2017)

Programovací jazyky

Mezi imperativní programovací jazyky se řadí COBOL, Pascal, Basic a C.

Neprocedurální programovací jazyky

Neprocedurální jazyky jsou faktograficky orientované programovací jazyky. Programy napsané v těchto jazycích specifikují, co se má udělat, a neuvádějí přesně, jak se má výsledek vyhodnotit. V neprocedurálním programovacím jazyce programátor specifikuje, co se má udělat, ale nezabývá se částí, jak se to má udělat. Je znám jako aplikační nebo funkcionální jazyk, protože pracuje s pomocí matematických funkcí. Sémantika neprocedurálního jazyka je ve srovnání s procedurálními jazyky jednodušší. (Panigrahi, 2022b)

Programovací jazyky

Mezi neprocedurální programovací jazyky se řadí LISP a Prolog.

3.3.3 Kompilované a interpretované

Kompilované jazyky potřebují pro svůj zdrojový kód překladač neboli kompilátor, který ho přeloží do strojového kódu, poté je kód teprve spustitelný. Interpretované jazyky nevyužívají překladač, ale používají speciální program, takzvaný interpret, který kód čte a spouští tím daný program. (Baeldung, 2023)

Rozdíl mezi interpretovaným a kompilovaným jazykem spočívá ve výsledku procesu interpretace nebo kompilace. Interpret vytváří výsledek programu, zatímco kompilátor vytváří program zapsaný v assembleru. Assembler pak výsledný program převede do binárního kódu. Jazyk assembleru se u jednotlivých počítačů liší v závislosti na jejich architektuře. Z toho vyplývá, že zkompilované programy lze spustit pouze na počítačích, které mají stejnou architekturu jako počítač, na kterém byly zkompilovány. (Murat, 2023)

V dnešní době se lze častěji setkat s tím, že jsou programovací jazyky kombinací obou. Překladač přeloží kód do „meziformy“, která se odborně nazývá „bytecode“, ten je pak předán interpretovi a ten již může kód spustit. (BYJU'S Exam Prep, November 11th, 2022)

Kompilované programovací jazyky

Kompilované programovací jazyky jsou převedeny přímo do strojového kódu pomocí překladače. Díky tomu bývají rychlejší a efektivnější než interpretované jazyky. Vývojářům také poskytují větší kontrolu nad hardwarovými aspekty, jako je správa paměti nebo využití procesoru. Zkompilovaný kód je však závislý na hardwaru. (Murat, 2023)

Zkompilovaný program není pro člověka čitelný, ale je ve strojovém jazyce specifickým pro danou architekturu/počítač. Vytvoření zkompilovaného programu vyžaduje několik kroků. Nejprve programátor pomocí vývojového prostředí nebo i obvyčejného textového editoru napíše zdrojový kód ve zvoleném programovacím jazyce, pokud je program komplikovanější, mohou být jeho části rozloženy mezi více souborů. Poté programátor program zkompiluje, tím se roztrídí a propojí jednotlivé moduly a vše se převede do strojového kódu, kterému počítač rozumí. (Baeldung, 2023)

Pro různé operační systémy a různé architektury téhož operačního systému je třeba provést samostatnou kompilaci. Zkompilované jazyky tedy nejsou multiplatformní. Často je ovšem možné program překompilovat tak, aby běžel na různých platformách. (Sarangi, 2023)

Kompilované jazyky potřebují „build“, což znamená, že je nejdříve potřeba je ručně zkompileovat. Pokaždé, když je potřeba provést nějakou změnu v kódu, musí se celý program znovu přestavět respektive, udělat nový build programu. (Sarangi, 2023)

Výhody a nevýhody

Mezi výhody kompilovaných jazyků patří to, že programy, které jsou zkompileované do nativního strojového kódu bývají rychlejší než interpretovaný kód. Je to proto, že kód není třeba kompilovat za běhu programu. Kompilátor během kompilace poskytuje seznam všech chyb kompilace. Vývojář je pak schopen chyby opravit a kód znovu spustit. (Murat, 2023)

Mezi nevýhody lze zařadit potřebu dodatečného času k dokončení celé kompilace před spuštěním nebo závislost strojového kódu na platformě. Po kompilaci programu do strojového kódu není možné program dále upravovat. Pokud je upraven, je nutné ho znovu zkompileovat. (Murat, 2023)

Programovací jazyky

Mezi čistě kompilované programovací jazyky patří C, C++, Rust, Go nebo Haskell. (Sarangi, 2023)

Interpretované programovací jazyky

V případě interpretovaného jazyka je zdrojový kód obvykle samotným programem. Programy tohoto typu, lze často označit jako skripty, vyžadují interpreta, který zanalyzuje příkazy v programu, a poté je provede. Někteří interpreti, jako například unix shells, přečtou a poté okamžitě provedou každý příkaz. Zatímco jiné, jako například Perl, analyzují celý skript před odesláním odpovídajících instrukcí strojového jazyka. Výhodou skriptu je, že je velmi dobře přenositelný. Nevýhodou je, že pokud interpret není k dispozici, program se vůbec nespustí. (Baeldung, 2023)

Just-in-time kompilace (JIT)

Tento typ kompilace je metoda pro zlepšení výkonu interpretovaných programů. Během provádění programu může být program zkompileován do nativního kódu, aby se zlepšil jeho výkon. Je také známa jako dynamická kompilace. (Larson, 2020)

JIT kompilace pak funguje na principu toho, že je zdrojový kód převeden do struktury podobné jazyku assembleru, jako příklad lze uvést bytecode v jazyce Java. Tento bytecode je pak převeden do strojového jazyka pouze tehdy, když aplikace potřebuje, aby byly požadované kódy převedeny. (Larson, 2020)

Dynamická kompilace má oproti statické kompilaci pár výhod. Při spuštění aplikací v jazyce Java nebo C# může běhové prostředí profilovat⁴ aplikaci během jejího běhu. To umožňuje generovat optimalizovanější kód. Pokud se chování aplikace během jejího běhu změní, může runtime (běhové prostředí) kód překompilovat. Mezi nevýhody lze zařadit zpoždění při spouštění. Proto některé JIT kompilátory kompilují pouze často používané části kódu. (Larson, 2020)

Výhody a nevýhody

Výhody interpretovaného jazyka je snazší ladění programu a většinou dobrá přenositelnost programu na jinou platformu. Navíc bývají obvykle flexibilnější a často nabízejí funkce, jako je dynamické typování a menší velikost programu. Protože interpreti sami spouští zdrojový kód programu, je samotný kód nezávislý na platformě.

Mezi nevýhody je možno zařadit povinnost využívat interpreta konkrétního programovacího jazyka a pomalejší běh programu. (Prepbytes, 2023)

Programovací jazyky

Mezi čistě interpretované programovací jazyky patří Python, Ruby nebo JavaScript.

3.3.4 Strukturované programovací jazyky

Strukturované programování neboli modulární programování je programovací paradigma, které usnadňuje tvorbu programů s čitelným kódem a opakovaně použitelnými komponentami. Všechny moderní programovací jazyky podporují strukturované programování, ale mechanismy podpory, stejně jako syntaxe programovacích jazyků, se liší. (Nolle, 2023)

⁴ Profilování – sledování programu za běhu, přičemž získáváme různé údaje o jeho chování, díky nimž můžeme určit výkonnost jednotlivých částí

Strukturované programování podporuje rozdělení aplikačního programu do hierarchie modulů nebo autonomních prvků, které mohou obsahovat další takové prvky. V rámci každého prvku může být kód dále strukturován pomocí bloků příbuzné logiky, které mají zlepšit čitelnost a udržitelnost. (Nolle, 2023)

Strukturované programování má řadu výhod. Například podporuje implementaci shora dolů, což zlepšuje čitelnost i udržitelnost kódu. Strukturované programování také podporuje opakované použití kódu, protože i interní moduly lze vyjmout a osamostatnit, popsat v adresářích a odkazovat na ně v mnoha dalších aplikacích. (Nolle, 2023)

Programovací jazyky

Mezi strukturované programovací jazyky se řadí Pascal, C, C++ a Ada. Mezi strukturované jazyky můžeme řadit i Javu nebo C#, ty ale nelze zařadit pouze do jedné kategorie.

3.3.5 Objektově-orientované programovací jazyky

Objektově orientované programování, zkráceně OOP, je paradigma v programování, které se opírá o koncept tříd a objektů. Používá se ke strukturování programů do jednoduchých, opakovaně použitelných částí kódu, které se nazývají třídy. (BasuMallick, 2022)

V objektově orientovaném programování je velmi důležité znát spoustu základních pojmů a struktur, bez kterých se při programování neobejdeme. Je tedy pro pochopení úplného laika poměrně složité.

Dva nejzákladnější pojmy, které se v objektově orientovaném programování objevují jsou třída a objekt. Zatímco třída je programátorem definované rozložení nebo blueprint⁵ objektu, který popisuje, jak bude určitý druh objektu vypadat. Objekt je jednotlivá instance třídy, která obsahuje data a metody. (Nishad, 2023)

V objektovém programování jsou čtyři základní pilíře: dědičnost, zapouzdření, abstrakce a polymorfismus. (Herrity, 2023)

Zapouzdření

Zapouzdření neboli zabalení dat a metod do jedné komponenty. Zapouzdření atributů probíhá uvnitř objektu, ve kterém jsou uloženy všechny důležité informace a vnějšímu světu

⁵ Blueprint – návrh budoucí podoby

jsou zpřístupněny pouze vybrané informace. Jakmile je ze třídy vytvořena instance objektu, data a metody jsou zapouzdřeny přesně v tomto objektu. Zapouzdření skrývá vnitřní implementaci softwarového kódu uvnitř třídy a vnitřní data objektů uvnitř třídy. V praxi to znamená, že jednotlivým atributům se nastaví přístup, a to buď public, private nebo protected (veřejný, soukromý nebo chráněný). (Herrity, 2023)

Pokud je atribut veřejný, lze k této vlastnosti přistupovat i z jiných tříd. Pokud se jedná o atribut soukromý, přístup k těmto vlastnostem, potažmo metodám, mají pouze v té třídě samotné. Aby bylo možné se k těmto soukromým vlastnostem dostat, používají se speciální metody, které přístup povolí, pokud je to zapotřebí. Tyto metody se nazývají „setter“ a „getter“. (Herrity, 2023)

Abstrakce

Abstrakce je koncept objektově orientovaného programování, který ukazuje pouze podstatné atributy a skrývá nepotřebné informace. Hlavním účelem abstrakce je skrýt před uživateli nepotřebné detaily. Abstrakce se vztahuje k pohledu na něco, co zachovává obecnou formu nebo význam a zároveň omezuje přítomnost konkrétních detailů. Používá se k jednoduchému popisu věcí a pomáhá vytvářet abstraktní třídy, které deklarují jednu nebo více abstraktních metod. Pokud mají abstraktní metodu, je nutné ji implementovat. Z abstraktních tříd nelze vytvářet instance. (Herrity, 2023)

Dědičnost

Dědičnost umožňuje ostatním třídám dědit vlastnosti jiných tříd. Potomci dědí od rodičů všechny jejich atributy a chování. To umožňuje využívat stejné chování i v jiné třídě bez toho, aniž by bylo zapotřebí metody znovu psát. Samotnou funkčnost těchto metod lze přepisovat a měnit tím i jejich chování podle potřeby. (Herrity, 2023)

Velkou výhodou je, že lze vytvořit obecnou rodičovskou třídu, a pak podle potřeby od ní „oddědit“ další třídy, které zdědí metody a její vlastnosti. Tímto se značně zjednoduší programování a potomek tak automaticky získá přístup ke všem metodám v rámci své rodičovské třídy, které jsou public nebo protected.

Polymorfismus

Polymorfismus znamená, že objekty jsou navrženy tak, aby sdílely chování. Pomocí dědičnosti mohou objekty přepsat sdílené rodičovské chování specifickým podřízeným chováním. Polymorfismus umožňuje, aby stejná metoda prováděla různá chování dvěma způsoby: přepisováním metod a přetěžováním metod. (Herrity, 2023)

Výhody a nevýhody

Objektové programování modeluje složité věci jako reprodukovatelné a jednoduché struktury. Pomocí zapouzdření vlastností ve třídách bezpečně chrání citlivé informace. Díky využití polymorfismu umožňuje specifické chování pro danou třídu. (Herrity, 2023)

Jednou z největších nevýhod se udává složitější učení. V objektově orientovaném programování přináší spoustu nových pojmů, které mohou být zpočátku obtížné pochopit. Programátoři, kteří přecházejí z procedurálního programování, musí poměrně výrazně změnit své myšlení, aby si osvojili to objektově orientované myšlení. Navíc OOP výrazně zvyšuje spotřebu paměti nebo potencionální vyšší požadavky na výkon. (Rini, 2023)

Programovací jazyky

Mezi objektově orientované programovací jazyky se řadí Java, C++ a Python. Jazyky OOP se nutně nemusí omezovat pouze na objektové programování. Některé programovací jazyky umožňují jak objektový, tak procedurální styl programování. (Herrity, 2023)

3.3.6 Typová kontrola

Typová kontrola určuje, jakým způsobem se deklarují, používají a spravují datové typy v programovacím jazyce. Každý programovací jazyk má jiný systém, který je zabudovaný do překladače, kompilátoru nebo je součástí běhové času programu. Kontrola typů je nezbytná pro minimalizaci chyb při spuštění programu.

Statically a dynamicky typované

Ve staticky typovaných programovacích jazycích dochází ke kontrole typu v době kompilace. Při kompilaci se zdrojový kód převede do strojově čitelného formátu. To znamená, že před kompilací zdrojového kódu musí být určen typ přiřazený ke každé proměnné. Protože typová kontrola probíhá během kompilace, je obvykle rychlejší odhalení typových chyb v programech. (Baeldung, 2023)

Naopak dynamicky typované programovací jazyky kontrolují datové typy při spuštění nebo až za běhu programu. U dynamického typování není vyžadováno explicitní uvedení datového typu při deklaraci proměnné. Systém jazyka je totiž schopen zjistit typ proměnné až za běhu. (Baeldung, 2023)

Silně a slabě typované

Druhým rozdělením v rámci typové kontroly je silné a slabé typování.

U silně typovaných programovacích jazyků musí mít proměnné svůj datový typ, který nelze měnit. Proměnnou nelze použít jako operand v žádné operaci, pokud její typ není kompatibilní. Jednoduše řečeno, nelze sčítat číslo s textem a naopak. V kódu se pak sice nachází více textu, za to však silné typování usnadňuje pochopení kódu, protože se v něm nachází méně skrytého chování. (TekTutorialsHub, 2022)

Oproti tomu ve slabě typovaných programovacích jazycích se mění datový typ proměnné v závislosti na jejím použití. Při běhu programu může nastat v případě potřeby implicitní typová konverze. Jako operand operace dělení lze předat textový řetězec a v průběhu programu se převede na číslo. To ovšem může způsobit chybu nebo také neplatný výsledek, pokud nelze převést obsah na platné číslo. (TekTutorialsHub, 2022)

3.4 Aplikační využití

Důležitým aspektem při výběru jazyka je i jeho vhodnost pro psaní konkrétních aplikací. Některé programovací jazyky jsou obecně využívány napříč různými obory a jiné jsou čistě specializované pro konkrétní oblasti, proto je velmi důležité znát jejich přednosti. Ačkoliv lze teoreticky jakýkoliv jazyk využít na všechno, bývá běžnou praxí používat konkrétní jazyky pro danou problematiku.

3.4.1 Desktopové aplikace

Aplikace pro počítače je software, který je určen pro spuštění na počítačích a využívá systémové prostředky k provedení svých operací. Tyto aplikace běží na různých operačních systémech jako je Microsoft Windows, Linux nebo MacOS. Poskytují samostatné uživatelské rozhraní a mohou nabízet širokou škálu funkcí. Na rozdíl od webových nebo mobilních aplikací se desktopové aplikace stahují a instalují do počítače uživatele, což jim umožňuje fungovat „offline“ bez závislosti na nepřetržitém připojení k internetu. Je nutné

brát v potaz, jaký programovací jazyk je vhodný, pro jaký operační systém. Mezi nejoblíbenější jazyky pro vývoj desktopových aplikací patří C#, Java nebo Python. (Patadiya, 2023)

3.4.2 Webové aplikace

Vývoj webových aplikací spočívá ve vytváření a údržbě webových stránek. Rozděluje se na dvě kategorie – front-end a back-end vývoj. Je zapotřebí si vybrat, zdali se studenti mají věnovat vzhledu webové aplikace nebo její funkčnosti na serveru. Webovou aplikací lze nazvat aplikační program, který je uložený na vzdáleném serveru a poskytovaný prostřednictvím protokolu HTTP přes rozhraní prohlížeče. Webové aplikace využívají kombinaci skriptů na straně serveru (PHP a ASP), které se starají o ukládání a načítání informací, a skriptů na straně klienta (JavaScript a HTML), které informace prezentují uživatelům. (StackPath, 2023)

3.4.3 Mobilní aplikace

Vývoj mobilních aplikací je v dnešní době velmi populární a taky žádaný z hlediska pracovních nabídek. Je zde zapotřebí brát v úvahu i operační systém, a to konkrétně Android nebo iOS. Oproti desktopovým aplikacím mají různá omezení, nejčastěji se jedná o hardwarová. Mobilní aplikace rozdělujeme na dva druhy – nativní a hybridní.

Nativní aplikace jsou vyvíjeny přímo pro konkrétní OS, protože jsou vyvíjeny specifickým jazykem pro něj. Pro Android aplikace se jedná o Javu nebo Kotlin, pro iOS je to Objective-C nebo Swift. Hybridní aplikace jsou kombinací webové a nativní aplikace a mohou využívat různé funkce zařízení, na kterém jsou nainstalovány. Tyto aplikace jsou multiplatformní a lze je najít pouze v obchodech jako je Google Play nebo App Store. (Kapias, 2023)

3.4.4 Datová analýza a umělá inteligence

Při práci v datové analýze se často pracuje s matematickými a statickými funkcemi pro manipulaci s daty, jejich analýzu a získání informací. Tato věda je se zabývá od strojového učení, analýzu sítí, zpracování přirozeného jazyka až po geoprostorovou analýzu⁶. Je nutno brát v potaz, že se pracuje s velkým množstvím dat, a proto je zapotřebí programovací jazyk,

⁶ Geoprostorová analýza – měří vlastnosti a vztahy v rámci zkoumaného území

který má vysokou produktivitu a výkon. (Luna, 2023) Analýza dat kontroluje, čistí, transformuje a modeluje data s cílem získat poznatky a podpořit rozhodování. S rostoucím objemem dat je jejich zkoumání obtížnější a časově náročnější. Analýza dat pomocí umělé inteligence využívá různé techniky umělé inteligence k získání cenných poznatků z velkého množství dat. Jako tři nejznámější techniky se udávají: algoritmy strojového učení, hluboké učení a zpracování přirozeného jazyka. (Luzmo team, 2023)

Mezi nejvyžívanější programovací jazyky v této oblasti patří Python, Java, Scala nebo C/C++. (Luna, 2023)

4 Výuka programovacích jazyků na SŠ

Tato kapitola je zaměřena na vyučování programovacích jazyků dle rámcového vzdělávacího programu středního odborného vzdělávání (RVP SOV). Zde je zapotřebí zmínit, co tyto programy obsahují, a jaký kladou důraz na výuku v této oblasti. Dále je zde definován pojem ŠVP a RVP.

4.1 Rámcové vzdělávací programy a školní vzdělávací programy

„Rámcové vzdělávací programy (RVP) jsou centrálně zpracovanými pedagogickými dokumenty, které schvaluje a vydává Ministerstvo školství, mládeže a tělovýchovy, pro každý obor vzdělání je vydán samostatný RVP.“ (NPI ČR, 2023)

RVP obecně specifikují jednotlivé požadavky na vzdělávání pro jednotlivé stupně a obory na každé škole. Určují nejen vzdělávací cíle, které by měli být naplněny, ale také stanovují oblasti vzdělávání a vymezují formy vzdělávání. Všechny tyto požadavky jsou zavazující pro každou školu, která podle nich sestavuje své vlastní Školní vzdělávací programy (ŠVP). (NPI ČR, 2023)

ŠVP si připravuje každá škola sama na základě příslušného RVP, a to pro každý obor vzdělávání, podle kterého se pak následně realizuje výuka. ŠVP zohledňuje vzdělávací podmínky dané školy a schvaluje jej ředitel školy. Ředitel školy také zodpovídá za soulad s příslušným RVP. Název ŠVP pro každý obor si může určit škola sama, název by měl ovšem co nejvíce odpovídat zaměření oboru. (NPI ČR, 2023)

4.2 RVP středního odborného vzdělávání

RVP proběhlo v srpnu 2023 velkou změnou v oblasti „Vzdělávání v informačních a v komunikačních technologiích“. RVP středního odborného vzdělávání se mění od 1. září 2023. Změna se týká třech složek kurikula, a to konkrétně: zavedení nového pojetí informatického vzdělávání, nové koncepce digitální koncepce a průřezového tématu Člověk a digitální svět. Dobrovolně mohou školy tuto změnu zapracovat do svých ŠVP již od září 2023, povinnost vyučovat podle této změny platí až od září 2025. (MŠMT ČR & NPI ČR, 2023)

„Změna ze srpna 2023 RVP SOV v oblasti ICT se týká zavedení v návaznosti na revizi RVP základního vzdělávání a aktualizaci RVP a pro gymnázia, jakož i dynamický vývoj v oblasti ICT, nového pojetí Informatického vzdělávání (s nově koncipovanou Digitální kompetencí a průřezovým tématem Člověk a digitální svět) ve všeobecně vzdělávací složce kurikula.“ (MŠMT ČR, 2023)

RVP středního odborného vzdělávání pro obor Informační technologie obecně specifikuje učivo a výsledky vzdělávání. Pokud se jedná o Informatické vzdělávání, ve kterém se právě nachází i výuka programovacích jazyků, učivo obsahuje: *„základní koncepce tvorby programů (např. proměnná a datový typ, řídicí příkazy, cykly), návrh algoritmů a datových struktur, zápis algoritmu vhodnou formou (např. blokové schéma, přirozené a formální jazyky, skriptovací a programovací jazyk) a využívání hotových komponent.“ (MŠMT ČR, 2023)*

4.3 ŠVP středního odborného vzdělávání

Výuka programování probíhá napříč všemi ročníky na středních odborných školách pod různými názvy předmětů. Každá škola, která vychází z RVP v oblasti informatiky a ICT, nabízí jiné programovací jazyky a jiný pohled na výuku. V některých školách je výuka programování shrnuta pouze do jednoho předmětu, jinde se nachází předmětů více, ty jsou poté rozděleny podle aplikačních oblastí. Vzhledem k tomu, že nejnovější změny z RVP ještě nejsou aplikovány, informace jsou zpracovány z ŠVP, které v sobě nemají zapracované nové změny. Tato práce se primárně soustředí na ŠVP středních odborných škol, které mají obor Informační technologie (18-20-M/01), konkrétně jejich výuku programovacích jazyků.

Při zařazení programovacího jazyka do ŠVP je nutno brát ohled na spoustu faktorů, které mohou ovlivnit jeho vhodnost.

Časová dotace

Prvním důležitým faktorem je časová dotace, kterou jednotlivé školy nabízí. Některé školy nabízí výuku programování již od prvního ročníku, jiné se k tomuto tématu dostanou až ve třetím ročníku. Nejčastěji školy nabízí se dvě hodiny týdně, občas tři hodiny, a to pouze jen v určitém ročníku. Vzhledem k tomu, že je výuka programování velmi náročná, je potřeba zajistit dostatečnou časovou dotaci. Od toho se odráží množství probrané látky za jeden

ročník a také výběr samotného jazyka. Pokud školy vyberou náročný programovací jazyk, studenti se mohou naučit mnohem méně věcí, než kdyby byl zvolen mnohem jednodušší. Pokud učitel stráví velké množství času při vysvětlování samotného zápisu kódu nebo jeho funkčnosti, nemůže probrat základní témata, která s programováním souvisí.

Znalosti

V podstatě všechny ŠVP v programování začínají úvodem do algoritmizace, kde jsou studentům nejdříve představeny základní pojmy a charakteristiky. A od tohoto se jednotlivé programy liší. Zatímco jedna škola počítá i s úvodem do vývojového prostředí, jiná škola začíná se základními prvky programování jako jsou proměnné, datové typy a operátory. Poté často přechází na metody, podmínky, cykly a základní datovou strukturu – pole. V téměř žádném ŠVP není přesně specifikován programovací jazyk.

Zajímavé ovšem je, že v některých ŠVP není vůbec řečeno ani jaké programovací paradigma má učitel zvolit. Někde je program natolik obecně napsaný, že je téměř jedno, v jakém programovacím jazyce se studenti budou učit. Oproti tomu v jiných programech je paradigma přesně řečeno a nejčastěji se objevují dvě – strukturované nebo objektově orientované. Lze tedy říci, že pohled na tuto problematiku je velmi rozdílný.

V dalších navazujících ročnících se objevují typická témata, se kterými se v počátcích programování seznamuje každý. Studenti se učí jednotlivé kolekce, datové struktury, datové proudy, návrhové vzory, vlákna, práce se soubory a regulární výrazy. V posledních ročnících se často objevuje provázání s dalšími předměty jako je propojení vlastního kódu s databází, vytvoření síťové nebo webové aplikace.

Výuka na ZŠ

Jedním z dalších faktorů může být i samotná výuka programování na základní škole. Některé základní školy programování v rámci informatiky učí, některé nikoliv. Je tedy velký rozdíl, zdali žáci už s nějakými znalostmi přichází nebo ne. Pokud se žáci na ZŠ setkají například pouze s blokovým programováním, je nejlepší možností na tyto znalosti navázat už v textových programovacích jazycích. Bohužel na středních školách se učitelé setkávají s velkými rozptyly a je nutné s tímto počítat a jejich znalosti vyrovnat.

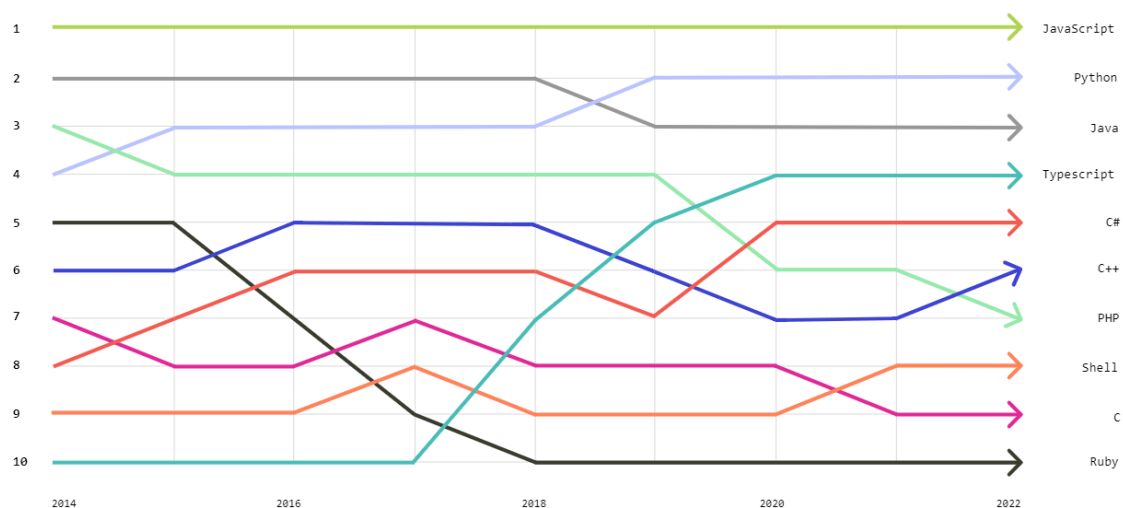
Textové programovací jazyky

Právě textové programovací jazyky by měly být předmětem pro rozhodování. Studenti by měli být dostatečně připravení na to rozvíjet své programátorské schopnosti již v plně využitelných programovacích jazycích a je zcela zbytečné se zabývat blokovými jazyky, se kterými se právě často setkávají na základních školách. Ačkoliv blokové jazyky dokážou rozvíjet algoritmické myšlení, zároveň velmi zjednodušují syntaktická pravidla zápisu zdrojového kódu. Studenti díky textovým jazykům se naučí základní syntaxi a prvky programování, které pak mohou využít v rámci praxí nebo v pracovním životě.

5 Výběr programovacích jazyků

V této kapitole jsou vybrány jednotlivé programovací jazyky, které jsou momentálně nejvíce využívány programátory po celém světě. Každý programovací jazyk je tu specifikován nejen z hlediska funkcionality a jeho využití v oblastech IT, ale je zde také stručně popsána syntaxe a sémantika.

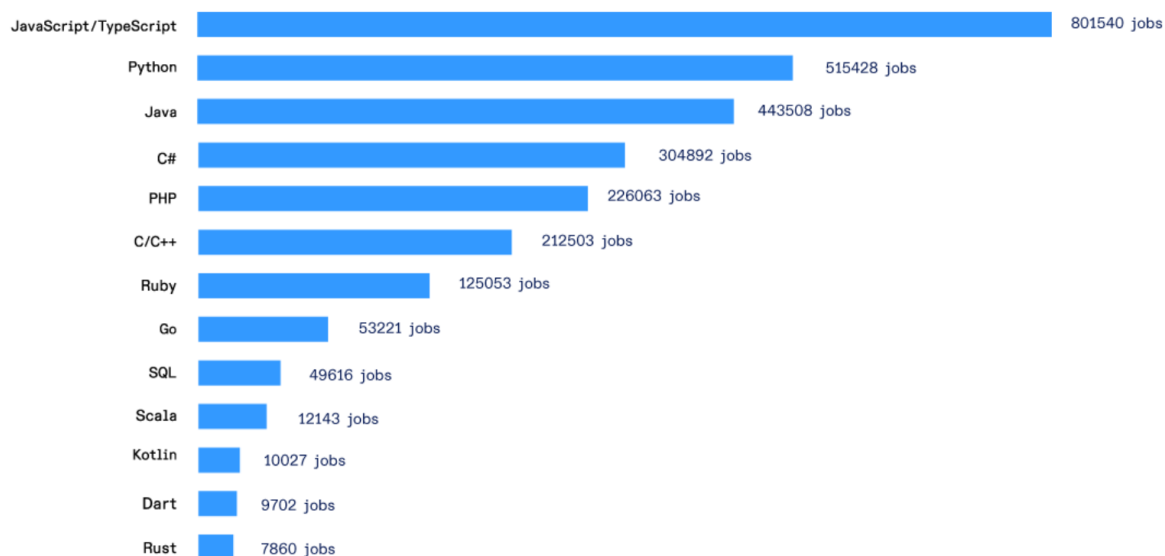
Například GitHub⁷, který je velmi hojně využíván programátory i firmami po celém světě, každoročně sdílí nejvíce využívané jazyky v rámci projektů, které jsou na něm uloženy. Tento graf lze velmi dobře použít jako vzhled do popularity jazyků a může pomoci při výběru.



Obrázek 1 - Vývoj nejpoužívanějších programovacích jazyků na GitHubu

Protože právě populární jazyky jsou velmi často vyhledávány i firmami, které hledají nové zaměstnance. Učitelé by měli vybrat takový jazyk, který umožní žákům získat zkušenosti, které by jim mohly pomoci i v pracovním životě.

⁷ GitHub – platforma pro verzovací systém Git, kde programátoři mohou sdílet své projekty



Obrázek 2 - Nejžádanější programovací jazyky podle počtu pracovních míst

Pokud se porovnají tyto dva grafy, lze pozorovat, že oblíbené programovací jazyky jsou zároveň těmi, které jsou vyžadovány po uchazečích v IT firmách. Pokud se studenti naučí hned od začátku v jednom z nich pracovat, mohou získat brzký vhled do problematiky daného jazyka. Je ovšem nutné brát v potaz i ten fakt, že pracovní pozice se neustále mění a nelze to brát jako jediný faktor při rozhodování.

5.1 Jazyk C

Programovací jazyk C je procedurální, kompilovaný a strukturovaný jazyk, který byl vyvinutý pro specifický operační systém UNIX. Ten stejně jako i většina programů, které běží na tomto systému, je napsán v jazyce C. (Kernighan, 2019) Dennis Ritchie vytvořil jazyk C v roce 1972 ve společnosti AT&T, stal se tak nástupcem jazyka B. Jazyk C je navržen tak, aby překonal problémy, se kterými se potýkaly programovací jazyky jako BASIC, BPCL nebo právě B. Dnes se využívá převážně pro tvorbu kompilátorů, operačních systémů a mikropočítačů. (Simplilearn, 2023)

„C je relativně „nízkourovňový“ jazyk. To není myšleno pejorativně; tím chceme říci, že C pracuje se stejnými objekty jako většina počítačů, jmenovitě se znaky, čísly a adresami. S tím vším je možnost pracovat pomocí aritmetických a logických operátorů implementovaných skutečnými počítači.“ (Kernighan, 2019)

Jazyk C také poskytuje operace, které přesně odpovídají instrukcím zabudovaným v počítači, takže programy mají větší předpoklad k tomu běžet efektivněji. Aplikační programy se na něj spoléhají v souvislosti se vstupem/výstupem, správou úložišť a řadu dalších služeb, proto si nemůže operační systém dovolit být pomalý. (King, 2008)

5.1.1 Funkcionalita

Zdrojový kód se nachází v souboru s příponou .c a .h. Program by měl být převeden do podoby, kterou může stroj spustit. To obvykle zahrnuje tři kroky.

Nejdříve proběhne „předzpracování“. Program se nejprve předá preprocesoru, který se řídí příkazy začínajícími znakem # (tzv. direktivami). Preprocesor je něco jako editor; může do programu přidávat věci a provádět úpravy. Poté upravený program přejde do překladače, který jej přeloží do strojových instrukcí (objektového kódu). V posledním kroku linker⁸ spojí objektový kód vytvořený kompilátorem s dalším potřebným kódem, aby vznikl kompletní spustitelný program. Tento dodatečný kód zahrnuje funkce z knihoven (například printf), které se v programu použijí. (King, 2008)

5.1.2 Využití C

V programovacím jazyce C bylo napsáno jádro systému UNIX, aplikace operačního systému Microsoft Windows a velká část operačního systému Android. Aplikace v jazyce C se běžně používají při tvorbě 3D videí, protože je potřeba během tvorby zpracovávat velké množství dat a provádět mnoho výpočtů za sekundu, musí být tyto aplikace extrémně rychlé a efektivní. (Ravikiran, 2023)

Navíc programovací jazyk C vedl k vytvoření mnoha dalších jazyků jako je C++, C#, Python, Java, PHP a další. Každý jazyk využívá C v různé míře. Zatímco Python používá C jako standardní knihovny, C++ a PHP potřebují jazyk C pro svou syntaxi a architekturu řízení. Programovací jazyk C je doporučeným jazykem pro vytváření ovladačů a aplikací vestavěných systémů. Dostupnost hardwarových rozhraní API na strojové úrovni, stejně jako přítomnost překladačů jazyka C a dynamické přidělování paměti činí tento jazyk nejoblíbenějším. (Ravikiran, 2023)

⁸ Linker – propojovací program sloužící ke spojení samostatně přeložených modulů a knihoven do funkčního celku

5.1.3 Syntaxe a sémantika

Programovací jazyk C je staticky a slabě typovaný. Každý příkaz končí středníkem a bloky příkazů se ohraničují složenými závorkami. Má pevně daná klíčová slova, umožňuje klasické aritmetické, bitové i logické operátory. Funkce se nesmí jmenovat stejně a mohou volat sami sebe, takže podporují rekurzi. Názvy funkcí i proměnných začínají podle napsaného pravidla malým písmenem. Obsah pole nelze porovnávat pomocí jediného vestavěného operátoru, ale prvky se musí porovnávat jednotlivě. Složité funkce jako je práce s datovými toky, manipulace s řetězci a matematické funkce jsou zahrnuty v knihovnách.

5.2 Jazyk C++

Programovací jazyk C++ je verze tradičního programovacího jazyka C rozšířená o objektově orientované programování a další koncepty. Byl vytvořen významným počítačovým vědcem Bjornem Stroustrupem jako multiplatformní vylepšení jazyka C, aby poskytl vývojářům vyšší míru kontroly nad pamětí a systémovými prostředky. C++ je velmi užitečný jazyk, který je využíván nejen při samotném programování počítačů, ale i při výuce nových programátorů při objektově orientovaném programování. (Rouse, 2021)

Nepodporuje však pouze objektové, ale také procedurální a funkcionální programování. Programovací jazyk C++ je navíc přenositelný mezi platformami, takže umožňuje vývojářům velmi snadno vytvářet programy, které mohou běžet na různých operačních systémech nebo platformách. Přestože se jedná o vysokoúrovňový jazyk, lze jej díky jeho úzkému vztahu ke strojovému jazyku používat i pro nízkoúrovňovou manipulaci. (Rouse, 2021)

5.2.1 Funkcionalita

Zdrojový kód se nachází v souboru s příponou .cpp a .h. Překladač C++ překládá zdrojový kód do strojového jazyka a ukládá jej na disk s příponou souboru .o. Linker pak tento soubor s objektovým kódem propojí se soubory standardních knihoven, které kód programu vyžaduje, a tak nakonec vytvoří spustitelný soubor, který je opět uložen na disku. Zatímco se pokouší spustit a vykonat soubor s kódem v jazyce C++, spustitelný soubor se načte z disku do paměti a procesor pak program vykoná po jedné instrukci. (Jain, 2022)

5.2.2 Využití C++

Jazyk C++ se stal velmi oblíbeným jazykem pro vytváření rychlých a spolehlivých systémů. Díky své univerzálnosti se výborně hodí pro komplexní aplikace, vestavěné systémy a zařízení internetu věcí. (IoT)⁹. Často se využívá v softwarovém inženýrství, byly v něm vytvořeny aplikace jako je Netflix nebo Amazon. (Coursera, 2023b)

Mimo jiné se jazyk C++ doporučuje pro návrh GUI¹⁰ nebo pro vyvíjení her, herních konzolí nebo i VR¹¹. (Coursera, 2023b)

5.2.3 Syntaxe a sémantika

Syntaxe C++ je velmi podobná syntaxi jazyka C. Jazyk C++ podporuje funkce, třídy, objekty a šablony¹². Protože je to jazyk statický a silně typovaný vyžaduje, aby všechny proměnné měly konkrétní typ, a to buď explicitně deklarovaný programátorem, nebo vyvolaný kompilátorem. Navíc nabízí více než 35 operátorů, které pokrývají základní aritmetiku, práci s bity, indirekci, logické operace a další. Téměř všechny operátory lze přetížit pro uživatelem definované typy. Také podporuje anonymní metody a lepší práci s výjimkami.

5.3 Jazyk C#

C# je objektově orientovaný a kompilovaný programovací jazyk od společnosti Microsoft. C# kombinuje výpočetní výkon jazyka C++ a jednoduchost jazyka Visual Basic a prostředí společnosti Microsoft pro událostmi řízené programování. (Jalli, 2022)

5.3.1 Funkcionalita

Každý program v jazyce C# běží na pozadí prostřednictvím rámce Microsoft .NET, což je běhová platforma pro jazyk C# a další jazyky, jako je Visual Basic. (Jalli, 2022) Programy se spouští ve virtuální spouštěcím systému zvaném common language runtime (CLR) a sadě knihoven tříd. CLR provede kompilaci JIT (Just-In-Time), která převede kód CIL do strojového kódu. Zároveň spravuje garbage collection, zpracování výjimek a správu prostředků. (Wagner, 2023)

⁹ IoT – anglická zkratka pro internet věcí

¹⁰ GUI – anglická zkratka pro grafické uživatelské rozhraní

¹¹ VR – virtuální realita

¹² Šablony – je konstrukce, která vygeneruje běžný typ nebo funkci v době kompilace na základě argumentů, které uživatel poskytuje pro parametry šablony

.NET Framework

.Net Framework je platforma pro vývoj softwaru vyvinutá společností Microsoft pro vytváření a spouštění aplikací pro systém Windows. Framework .NET se skládá z vývojářských nástrojů, programovacích jazyků a knihoven pro vytváření desktopových, webových i mobilních aplikací. Používá se také k vytváření webových stránek, webových služeb a her. (Thompson, 2023)

Architektura .NET je programovací model, který poskytuje prostředí pro spouštění a integraci s různými programovacími jazyky pro jednoduchý vývoj. Skládá se z knihoven tříd a opakovaně použitelných komponent. (Thompson, 2023)

5.3.2 Využití C#

Programovací jazyk C# je používán k vytváření různých typů programů a aplikací. Nejčastěji se využívá pro vývoj webových aplikací, aplikace pro Windows a vývoj her. Mezi vývojáři her je C# preferovaným jazykem. Herní engine Unity – jeden z nejpobulárnějších herních enginů současnosti – je vytvořen pomocí C++ a C#.

5.3.3 Syntaxe a sémantika

Jazyk C# je staticky a silně typovaný programovací jazyk a je také case sensitivní. Základní syntaxe C# je podobná syntaxi C a C++. Pro bloky příkazů se používají složené závorky a každý příkaz musí končit středníkem. Metody se seskupují do třídy a třídy do jmenných prostorů (namespace). Proměnné můžeme porovnávat přes dvě ==. Hranatými závorkami se označuje pole. C# podporuje veškeré vlastnosti OOP, stejně jako používání externích knihoven za pomoci správce balíčků nugget.

5.4 Java

Java je jeden z nejvíce rozšířených programovacích jazyků. Jedná se o open source objektově orientovaný, strukturovaný a imperativní jazyk, který vznikl v roce 1995 společností Sun Microsystems. Javu lze považovat za interpretovaný i kompilovaný jazyk. Jeden z hlavních důvodů vzniku byla potřeba „přenositelnosti“ programu, protože v 90. letech minulého století se začaly počítače hojně rozšiřovat. Vznikla tím tak potřeba vytvořit jazyk, u kterého by nemuselo být nutné při každém přenosu na jiný počítač, program znovu

kompilovat. A tak vznikl programovací jazyk Java, podle názvu kávy, kterou pili její tvůrci. (Roubalová, 2015)

Sami tvůrci jazyka pak udávají, že při vývoji se drželi pěti hlavních požadavků kódu, který by měl být: jednoduchý, objektově orientovaný a povědomý, robustní a bezpečný, nezávislý na platformě a přenositelný, výkonný a interpretovaný, vícevláknový a dynamický. (Oracle, 2023)

5.4.1 Funkcionalita

Zdrojový kód se nachází v souborech s příponou .java a je psán za pomoci Java Development Kitu (JDK), bytecode pak v souborech s příponou .class. Program se ze zdrojového kódu zkompile do bytecode, a ten je pak následně zpracován virtuálním strojem (JVM), který ho interpretuje. Díky tomu, že JVM může být nainstalovaný na mnoho operačních systémech, je možné stejný soubor .class spustit jak na operačním systému Windows, Linux nebo Mac OS. Některé virtuální stroje provádějí za běhu další kroky, které zvyšují výkon naší aplikace, například překompiluje často používané části kódu do nativního.

Bytecode

Bytecode je mezikód mezi zdrojovým a strojovým kódem. Jedná se o nízkoúrovňový kód, který je výsledkem kompilace zdrojového kódu a je napsán ve vysokoúrovňovém jazyce. Dále je pak zpracováván virtuálním strojem, jako je například Java Virtual Machine (JVM). (Satyabrata, 2020)

Bytecode je po překladu interpretem do strojového kódu čitelný pro počítač. Je zkompileován tak, aby ho mohla JVM spustit bez ohledu na operační systém. Označuje se též jako přenositelný kód. (Satyabrata, 2020)

Garbage collection

Garbage collection je v jazyce Java proces, při kterém programy provádějí automatickou správu paměti. Když programy běží v JVM, vytvářejí se objekty na haldě, což je část paměti vyhrazená pro program. Pokud již nějaké nepoužívané objekty nebudou potřeba, tyto objekty najde garbage collector a odstraní je, aby uvolnil paměť. Hlavním cílem garbage collection je tedy uvolnit paměť haldy za pomoci odstranění nedosažitelných objektů. (Agarwal, 2022)

5.4.2 Využití Javy

Java se převážně využívá na tvorbu podnikových systémů, mobilních a webových aplikací, zařízení Internetu věcí (IoT), her, distribuovaných a cloudových aplikací.

Mezi nejznámější aplikace, které byly vyvinuty v jazyce Java, je velmi populární hra Minecraft, hudební mobilní aplikace Spotify nebo webové aplikace jako je Netflix nebo eBay. (Olumide, 2023)

V programovacím jazyce Java mohou vývojáři snadno vytvářet desktopové aplikace pomocí nástrojů jako je rozhraní API – Swing, AWT nebo JavaFX. Java navíc nabízí příjemné využití OOP programování a je multiplatformní. (Kayasth, 2022)

Platforma Java

Platforma Java je prostředí nezávislé na platformě a může být o něco pomalejší než nativní kód. Nicméně díky pokrokům v technologiích překladačů a virtuálních strojů, se můžeme přiblížit k výkonu nativnímu kódu, aniž by byla ohrožena jeho přenositelnost. Tato platforma se od klasických platform liší tím, že se jedná pouze o softwarovou platformu, která běží nad jinými platformami založenými na hardwaru. Dělí se na dvě základní složky: Virtuální stroj Javy a rozhraní pro programování aplikací (API) Javy. (Oracle, 1995)

Virtuální stroj je základem platformy Java, interpretuje bytecode a je přenášen na různé hardwarové platformy (operační systémy počítače). Rozhraní API je pak rozsáhlá sbírka hotových softwarových komponentů, které poskytují mnoho užitečných funkcí. Je tvořena z knihoven příbuzných tříd a rozhraní, tyto knihovny se nazývají balíčky (packages). (Oracle, 1995)

5.4.3 Syntaxe a sémantika

Java je staticky a silně typovaná a má velmi podobnou syntaxi jako C a C++. Nicméně má méně nízkourovňových prostředků než tyto jazyky. Oproti tomu v Javě neexistují globální funkce ani proměnné, nepodporuje mnohonásobnou dědičnost nebo zde nejdou přetěžovat operátory. Je zde přetížený pouze operátor +, který se používá jak pro spojování řetězců, tak pro aritmetické sčítání. Java též podporuje generické programování a anonymní metody a je case sensitivní. Veškerý kód musí být uvnitř tříd a metod.

5.4.4 Groovy

Groovy je objektově orientovaný programovací jazyk vytvořený organizací Apache Software Foundation. Je kompatibilní s prostředím Java a používá se jako skriptovací jazyk. Podporuje funkcionální programování, metaprogramování, efektivní zpracování dat XML a JSON implementované prostřednictvím JVM. Groovy je velmi podobný syntaxi Java a lze jej implementovat přímo do kódu v jazyce Java a zvýšit tak jeho použitelnost. (Pedamkar, 2023)

5.4.5 Kotlin

Kotlin je univerzální, staticky typovaný programovací jazyk původně navržen pro JVM a Android, který kombinuje vlastnosti objektově orientovaného a funkcionálního programování. Dnes se využívá převážně pro vytváření mobilních aplikací Android. (Heller, 2022)

5.5 JavaScript a TypeScript

JavaScript je skriptovací funkcionální i imperativní a procedurální jazyk, který byl původně navržen pro vytváření dynamických a interaktivních webových stránek a aplikací. Programy v tomto jazyce se nazývají skripty. Mohou být napsány přímo v HTML webové stránky a často se spouštějí automaticky při načítání stránky. Skripty jsou poskytovány a spouštěny jako prostý text. Ke svému spuštění nepotřebují žádnou speciální přípravu ani kompilaci. (Kantor, 2007)

TypeScript je prototypový, kompilovaný, strukturovaný programovací jazyk, který vychází z jazyka JavaScript. Je považován nejen za jazyk, ale i za sadu nástrojů. Byl navržen designérem jazyka C# Andersem Hejlsbergem jako open source programovací jazyk, který má být vylepšeným JavaScriptem s funkcemi navíc. (Shubel, 2023)

Protože TypeScript vychází z jazyka JavaScript, snaží se ho vylepšit o funkce, které právě v JavaScriptu chybí nebo nejsou dostatečné. Jeho čtyři základní cíle jsou: zefektivnění vývoje a zavedení volitelných typů do jazyka JavaScript, dřívější odhalení chyb a implementace plánované funkce JavaScriptu. (Shubel, 2023)

5.5.1 Funkcionalita

JavaScript lze využívat nejen v prohlížeči, ale také na serveru nebo vlastně na jakémkoli zařízení, které má speciální program zvaný JavaScript engine. JavaScriptový engine je jednoduše počítačový program, který interpretuje kód JavaScriptu. (Christopher, 2023)

Nejoblíbenější je engine V8 prohlížeče Google Chrome. V8 společnosti Google pohání prohlížeč Google Chrome a Node.js, což je prostředí pro běh JavaScriptu na pozadí, které se používá k vytváření aplikací na straně serveru. (Christopher, 2023)

Jako jednu z největších nevýhod JavaScriptu lze považovat obtížné hledání a opravování chyb. Částečně je to způsobeno tím, že JavaScript je interpretovaný jazyk, a proto je třeba kód testovat, zdali je funkční. A v případě, že se chyba vyskytne, vývojáři musí chybu najít. (Shubel, 2023)

TypeScript pomáhá kontrolu chyb usnadnit, při kompilaci kódu TypeScript identifikuje syntaktické chyby a upozorní vývojáře vygenerováním chyb kompilace. TypeScript tak pomáhá uživatelům zachytit chyby již při kompilaci, místo aby čekali až do doby běhu. (Shubel, 2023)

5.5.2 Využití JavaScriptu a TypeScriptu

JavaScript se používá k vytváření dynamických webových stránek, webových a mobilních aplikací, her, webových serverů a back-endových¹³ infrastruktur. Vývojáři používají JS převážně k přidávání animací, vyskakovacích oken, vyhledávacích panelů, tlačítek, zvuku a videa, chatovacích widgetů a dalších interaktivních prvků na webové stránky. Pomocí JS lze také zajistit aktualizace stránky v reálném čase bez nutnosti jejího opětovného načítání. (Paruch, 2023)

JavaScript se používá k vytváření většiny her a aplikací na internetu. Protože JS běží přímo v prohlížeči, umožňuje uživatelům hrát hry a spouštět aplikace bez stahování dalšího softwaru. A protože se JavaScript kompiluje do strojového kódu v průběhu jeho spouštění, právě díky kompilaci just-in-time, jednoduché funkce a procesy se provádějí rychleji. (Paruch, 2023)

¹³ Back-end – vnitřní logika aplikace, například kompletní administrace obsahu webových stránek

V posledních letech vývojáři používají JavaScript k vytváření aplikací a systémů, které využívají umělou inteligenci. Různé knihovny JS, jako jsou TensorFlow.js a Brain.js, je možno použít k vytváření a trénování modelů strojového učení pro analýzu sentimentu a doporučovací systémy. TypeScript se využívá ve stejné oblasti jako JavaScript. Převážně tedy jako back-end pro webové aplikace. (Paruch, 2023)

5.5.3 Syntaxe a sémantika

JavaScript je dynamicky, slabě typovaný a case sensitivní jazyk. Na rozdíl od jazyka C, mohou whitespaces¹⁴ ve zdrojovém kódu přímo ovlivňovat sémantiku. Příkazy se ukončují mohou středníky. Proměnné v JavaScriptu nemají přiřazený žádný typ, takže lze do proměnné přiřazovat libovolného hodnoty a reference. Je zde pouze přetížený operátor +, stejně jako v Javě. Složené závorky dávají příkazy do bloků a jdou zde vytvořit i třídy. JavaScript je založen na prototypech, díky kterým lze opakovaně používat chování existujících objektů, které lze následně klonovat a rozšiřovat.

Oproti tomu TypeScript je staticky a silně typovaný jazyk. Syntakticky je velmi podobný jazyky JavaScript, ale umožňuje zadávat typy dat předávaných v kódu a má možnost hlásit chyby, pokud se typy neshodují.

¹⁴ Whitespaces – Bílé znaky jako je například mezera nebo tabulátor

5.6 PHP

PHP (Hypertext Preprocessor) je open-source skriptovací, procedurální a interpretovaný jazyk na straně serveru, který používá mnoho vývojářů pro vývoj webových stránek. Je to také univerzální jazyk, který se využívá k tvorbě mnoha projektů, včetně grafických uživatelských rozhraní (GUI). (Kolade, 2021)

Udává se, že PHP je jedním z nejjednodušších programovacích jazyků, avšak realita využívání tohoto jazyka je zcela odlišná. Díky tomu, že je PHP skriptovací jazyk, tak se může zdát velmi přímočarý, ale při komplexnějším projektu je nutností využít OOP, což dělá jazyk značně komplexnější než jiné objektově orientované jazyky. (Shashoua, 2021)

5.6.1 Funkcionalita

Interpret PHP nejprve převezme .php soubor, ve kterém se nachází zdrojový kód a vytvoří sadu tokenů. Poté zkontroluje, zdali skript odpovídá syntaktickým pravidlům, a pomocí tokenů sestaví abstraktní syntaktický strom (AST)¹⁵. Jako další interpret vezme strom AST a přeloží uzly AST do nízkoúrovňových opkódu¹⁶ Zend. Nakonec jsou tyto opkódy spuštěny na virtuální stroji Zend Engine. Výstupem jsou jednotlivé příkazy, které jsou zapsané v souboru, které se buď provedou v konzoli nebo ve webovém prohlížeči. (Patel, 2021)

Dnes se používá služba OPcache, která zlepšuje výkon PHP tím, že ukládá předkompilovaný bytecode skriptu do sdílené paměti, čímž odstraňuje nutnost načítat a analyzovat skripty PHP při každém požadavku. (Patel, 2021)

5.6.2 Využití PHP

Díky snadnému používání se jazyk PHP stal základním jazykem pro back-end vývojáře a full-stack¹⁷ vývojáře. Obsahuje také vestavěné bezpečnostní funkce, jako je šifrování dat a omezení přístupu, což odstraňuje potřebu samostatné práce na zajištění bezpečnosti webových stránek. Kód PHP navíc často pohání online formuláře. (Shashoua, 2021)

Jelikož je PHP skriptovací jazyk, využívá se také pro psaní jednoduchých skriptů, které lze spouštět například pomocí automatizovaného Daemona. Výše uvedené knihovny mají velmi

¹⁵ AST – anglická zkratka pro abstraktní syntaktický strom, stromová reprezentace struktury zdrojového kódu zapsaného ve formálním jazyce

¹⁶ Opkód – číselný identifikátor jedné operace, kterou může následně provést virtuální stroj Zend

¹⁷ Full-stack vývojář – programuje nejen vnitřní logiku aplikace, ale i vzhled aplikace

rozmanitou funkčnost, kterou lze použít i na datovou analýzu, práci se soubory jako jsou excel, PDF a mnoho dalších. (Khan, 2021)

5.6.3 Syntaxe a sémantika

V porovnání s ostatními jazyky je syntaxe PHP velmi přímočará a mnohdy odpouští programátorům spoustu chyb. PHP je dynamicky a slabě typovaný jazyk a vychází ze syntaxe jazyka C, s výjimkami a vylepšeními pro jeho hlavní použití při vývoji webových stránek, kde se hojně využívá manipulace s řetězci, navíc podporuje OOP. Procesor PHP analyzuje pouze kód uvnitř jeho oddělovačů, každý příkaz musí být ukončen středníkem. Proměnné jazyka PHP musí mít předponu "\$".

5.7 Python

Python je univerzální vyšší a interpretovaný, procedurální, objektově orientovaný i strukturovaný programovací jazyk, jehož syntaxe je velmi blízká lidskému jazyku. Je dynamicky typovaný a využívá garbage collection. Má výkonné vestavěné funkce a knihovny pro analýzu dat a datovou vědu. Python byl vytvořen v roce 1991 nizozemským programátorem Guido van Rossumem, aby se zbavil omezení z programovacích jazyků ABC. Klíčovou roli v rychlém vývoji a rostoucí popularitě hraje to, že Python je open-source. Rozsáhlá komunita dobrovolných programátorů společně vyvinula tisíce modulů, které výrazně rozšířily možnosti Pythonu. Počátkem roku 2020 se stal Python jedním z nejrozšířenějších programovacích jazyků na světě. (Munro, 2023)

5.7.1 Funkcionalita

Python je z velké části interpretovaný jazyk, nikoli kompilovaný, i když kompilace je jedním z kroků. Kód jazyka Python je zapsán v souboru .py, ten je nejprve zkompilován do bytecodu, který se uloží do formátu .pyc nebo .pyo. Na místo toho, aby se instrukce prováděly na procesoru, se instrukce bytecodu provádějí na virtuálním stroji. (Karani, 2020)

Python Interpreter (VM)¹⁸ spouští kód v terminálu nebo v příkazovém řádku. Nejdříve přečte zdrojový kód, rozdělí jej na tokeny a identifikuje klíčová slova, identifikátory,

¹⁸ VM – virtuální stroj z anglické zkratky virtual machine

operátory a literály. Poté pro pochopení struktury a logiky kódu sestaví derivační strom¹⁹, a nakonec provádí program podle instrukcí definovaných v derivačním stromu. To zahrnuje provádění výpočtů, I/O²⁰ operací a volání funkcí. Navíc Python využívá stejně jako Java garbage collector pro automatickou správu paměti. (Vayola, 2016)

5.7.2 Využití Pythonu

Python se běžně používá k vývoji softwaru, automatizaci úloh, analýze dat a vizualizaci dat nebo i k vývoji webových stránek. Vzhledem k tomu, že se Python poměrně snadno učí, osvojilo si jej mnoho začínajících programátorů.

Python se převážně stal základem pro datové vědy a umožňuje datovým analytikům a dalším odborníkům používat tento jazyk k provádění složitých statistických výpočtů, vytváření vizualizací dat, sestavování algoritmů strojového učení, manipulace s daty a jejich analýze a k plnění dalších úkolů souvisejících s daty. (Coursera, 2023c)

5.7.3 Syntaxe a sémantika

Python je dynamicky a silně typovaný programovací jazyk. Jeho formátování zdrojového kódu by mělo být nenáročné a často používá anglická klíčová slova tam, kde jiné jazyky používají interpunkci. Na rozdíl od jiných jazyků nepoužívá k ohraničení bloků složené závorky, ale využívá odsazení (tabulátory). Zvětšení odsazení přichází po určitých příkazech, zmenšení odsazení znamená konec aktuálního bloku. Středníky za příkazy jsou povoleny, ale často se nevyužívají.

Metody jazyka Python mají explicitní parametr `self` pro přístup k datům instance, na rozdíl od implicitního `self` (nebo `this`) v některých jiných objektově orientovaných programovacích jazycích. Python také poskytuje metody, často nazývané *double under methods* (kvůli jejich názvům začínajícím a končícím dvojitém podtržítkem), které umožňují uživatelsky definovaným třídám upravit způsob, jakým se s nimi pracuje při nativních operacích včetně délky, porovnávání, při aritmetických operacích a konverzi typů. (Contributors, 2024)

¹⁹ Derivační strom – strom, který reprezentuje syntaktickou strukturu slovního řetězce podle formální gramatiky

²⁰ I/O – vstup a výstup z anglických slov – input/output

6 Porovnání programovacích jazyků

Tato kapitola je zaměřena na porovnání programovacích jazyků ze dvou hledisek – syntaxe a vhodnosti využití. V ukázkovém kódu v každém programovacím jazyce jsou vyobrazeny stejné základní příkazy, aby byly přehledně vidět jejich rozdíly a možnosti. Při výběru jazyka pro výuku lze do rozhodování přidat vícero faktorů, jako je čitelnost syntaxe, možnost využití OOP nebo třeba podpora od vývojářů ve formě dokumentace. Důležitá je třeba i samotná komunita u každého jazyka, která často vytváří různé přehledy, nápovědy nebo výukový materiál, který je volně dostupný na internetu. Nelze opomenout také možnost řešení chyb v kódu za pomoci debuggování²¹ nebo i nápovědu při psaní v různých IDE, které dodávají určitou pomoc během vývoje. Spousta těchto faktorů je nicméně subjektivní, a je pak na každém člověku, aby si vybral jazyk, který mu nejvíce vyhovuje.

6.1 Syntaxe programovacích jazyků

Syntaxe programovacích jazyků se liší z hlediska jeho vlastností, které jsou vypsány výše. V této podkapitole je každý jazyk porovnán v rámci celkového vzhledu kódu, deklarace proměnné, výpisu do konzole, podmínky, deklarace pole, cyklů for a foreach a tvorbou vlastních metod. Tyto prvky považuje RVP jako základ pro výuku. Díky ukázkám níže lze na první pohled zhodnotit nejen složitost zápisu, ale také přehlednost zdrojového kódu u každého jazyka. Bez znalostí základních příkazů v programování nelze kódu rozumět, proto jsou zde představeny.

Proměnná

V programování je proměnná výraz pro ukládání hodnot, které se často v běhu programu mění a lze je opakovaně používat. Proměnné mohou reprezentovat jakýkoliv typ dat, například celá čísla, textové řetězce, pole nebo třídy objektů. Záleží na programovacím jazyce, jakých datových typů může proměnná nabývat. (Rouse, 2022)

Proměnná má čtyři vlastnosti: identifikátor, umístění v paměti, datový typ a hodnotu. Pokud je jazyk dynamicky typovaný, nemusíme při deklaraci proměnné typ určovat. Nejčastěji se proměnná deklaruje v tomto pořadí:

```
datový_typ identifikátor = hodnota;
```

²¹ Debuggování – proces odstraňování chyb ve zdrojovém kódu

Výpis do konzole

Metoda pro výpis do konzole je z prvních metod, která se učí při výuce programování. V každém programovacím jazyce se tato metoda liší v názvu. Často se volají metody ze tříd, které podporují právě vstup nebo výstup datových proudů pro konzolové aplikace.

Podmínka

Podmínky v programování pomáhají takzvaně větvit program. Pokud podmínky nejsou v kódu využity, zdrojový kód se čte po řádcích a není možné, aby se nějaký kus kódu vynechal. Pokud jsou podmínky využity, lze zařídit, aby se část kódu provedla pouze tehdy, pokud je splněna stanovená podmínka. Jakmile podmínka splněna není, tato část kódu se neprovede a kód pokračuje dál. Jestliže podmínka splněna není, provede se ta část, která je schována pod klíčovým slovem `else`. Často základní kód pro podmínky vypadá takto:

```
if(logický výraz){  
    kód, který se provede, pokud je podmínka splněna  
}else{  
    kód, který se provede, pokud je podmínka nesplněna  
}
```

Cyklus

Cyklus neboli smyčka pomáhá k opakovanému provedení příkazů. Místo psaní stejného nebo podobného kódu několikrát po sobě, se tato část nahradí cyklem, který bude kód opakovat dle stanové podmínky.

V programování jsou tři základní cykly: `for`, `while` a `foreach`, ten se jako jediný nenachází v programovacím jazyce C. V ostatních jazycích existují cykly všechny. `For` cyklus je cyklus s takzvaným předem známým počtem opakování a cyklus se opakuje do té doby, dokud je splněna podmínka uvnitř hlavičky cyklu. Do těla cyklu poté se vkládá ten kód, který se má opakovat.

```
for (inicializace řídící proměnné; podmínka; inkrementace){  
    tělo cyklu - kód, který se bude opakovat  
}
```

Pole

Pole je datová struktura, do kterého se většinou ukládají prvky stejného datového typu. Výjimkou může být třeba JavaScript. Každý prvek má svůj vlastní index, na který se lze

v kódu odkazovat. Indexy v poli se u těchto vybraných programovacích jazyků číslují vždy od nuly. Prvky, které se v poli nachází, jsou v paměti uloženy za sebou. Pole se deklaruje obdobně jako proměnná, ale při deklaraci se využívají hranaté závorky [], aby bylo v kódu jasně poznat, že se jedná o tuto datovou strukturu.

```
datový_typ identifikátor[] = {hodnoty}
```

Metoda

Metody neboli funkce jsou bloky kódu, které se spustí tehdy, když jsou zavolány. Využívají se pro opakované použití v kódu, protože se uvnitř metody definuje kód, který lze mnohokrát využít. Pokud se jedná o metody u objektových jazyků dělí se na dva druhy: instanční a statické.

Statické metody nejsou vázány na třídu, což znamená, že jdou volat odkudkoli z kódu. Proto ve většině programovacích jazyků je metoda main, vstupní bod programu, statická. Oproti tomu jsou instanční metody vázány na samotnou třídu a voláme je přes instanci třídy. Ty jsou dostupné pouze v rámci dané instance třídy neboli toho objektu, který má v sobě tuto metodu definovanou.

Hlavička metody obsahuje přístupová práva, návratový typ, název metody a vstupní atributy metody. Přístupová práva jsou v jazycích nejčastěji public nebo private a definují viditelnost metod v programu. Návratový typ je buď void, metoda nám žádné hodnoty nevrací nebo specifikovaný datový typ. Pokud se jedná o dynamicky typovaný programovací jazyk, není potřeba datový typ určovat. Do metody předáváme vstupní data za pomoci vstupních atributů v kulatých závorkách v hlavičce metody. S těmito daty lze pak pracovat v rámci metody.

Do těla metody se píše kód, který bude proveden při volání metody. Pokud metoda má návratový typ, je potřeba, aby metoda obsahovala klíčové slovo return, které provádí návrat z metody se specifikovanou hodnotou.

```
přístupová_práva návratový_typ název(vstupní atributy) {  
    tělo metody  
}
```

Metoda se volá vždy za pomoci jejího názvu, a pokud má vstupní atributy, je důležité je vždy definovat datovým typem a identifikátorem.

6.1.1 C

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4.
5. int addNumber(int a, int b);
6.
7. int main(int argc, char ** argv)
8. {
9.     int number = 10;
10.    char name[5] = "Test";
11.
12.    printf("Number %i\r\n", number);
13.
14.    char* newName = malloc(strlen(name) + 4);
15.    if (newName) {
16.        strcpy_s(newName, sizeof newName, name);
17.        strcat_s(newName, sizeof newName, "test");
18.
19.        printf("%s", newName);
20.
21.        free(newName);
22.    }
23.    if (number >= 0) {
24.        printf("%s\r\n", "Greater or equal to zero");
25.    }
26.    else {
27.        printf("%s\r\n", "Less than zero");
28.    }
29.    char words[2][10] = {"Test", "Value"};
30.    int i;
31.    for (i = 0; i < sizeof(words) / sizeof(words[0]); i++)
32.    {
33.        printf("%s\r\n", words[i]);
34.    }
35.    for (i = 0; i < 10; i++) {
36.        printf("For index: %d\r\n", i);
37.    }
38.
39.    printf("%i", addNumber(13, 67));
40.    return 0;
41.}
42.
43. int addNumber(int a, int b) {
44.     return a + b;
45. }
```

Každý kód v programovacím jazyce C začíná klíčovým slovem **include**. V hranatých závorkách je specifikována hlavička knihovny, ze kterých se čerpají jednotlivé metody pro snadnější manipulaci. V tomto případě se jedná o standartní knihovnu jazyka C. Knihovna **stdio.h** slouží pro vstup a výstup, **stdlib.h** slouží pro práci s pamětí (alokace, dealokace) a **string.h** pro jednodušší práci se stringem.

Metoda musí být vždy definována před jejím použitím, na řádku pět je definována hlavička metody, pro pozdější využití v metodě main níže. Samozřejmě je možné přesunout celou metodu i na řádek pět, ale jedná se o jakýsi standard při programování v jazyce C. Na začátku se definuje pouze její hlavička, pro lepší přehled, co je v daném souboru možné volat za metody.

Metoda **main** je vstupním bodem programu, má většinou dva vstupní argumenty, **argc**, jedná se o počet argumentů, se kterým se daný program spouští například z konzole a **argv**, což je pole stringů, respektive pole argumentů, které se do programu zasílají. Metoda main zpravidla vrací číslo, které reprezentuje exit code daného procesu, kde 0 odpovídá úspěšnému ukončení procesu.

V jazyce C se převážně pracuje s ukazateli (pointery), které se značí při deklaraci proměnné pomocí hvězdičky (*). Textový řetězec lze definovat jako pole znaků (char) s fixní velikostí, přičemž pointer ukazuje na první znak daného řetězce. Pro vytvoření ukazatele na několik textových řetězců, jsou potřeba při deklaraci proměnné hvězdičky dvě, a tím vytvoříme pole textových řetězců. S tímto souvisí řádek deset, kde je vytvořen string druhým způsobem za pomoci hranatých závorek, do kterých se píše výše zmíněná velikost textového řetězce. Čísla se definují jako u většině programovacích jazyků pomocí klíčového slova **int**, názvu proměnné a následné přidání hodnoty.

Výpisy do konzole jsou nejjednodušší pomocí metody **printf** z knihovny stdio.h. Ta přijímá dva argumenty, a to formátovaný string, do kterého se za procento píše datový typ proměnné, která následuje jakožto druhý argument. Příkladem může být řádek dvanáct, kde je vypsáno číslo, tudíž je za procentem písmeno i, což znamená integer.

Pro alokování vlastní paměti pro textový řetězec, ke kterému se připojuje jiný řetězec, je nutné využít funkci **malloc**, viz řádek čtrnáct, který přijímá jakožto argument počet bajtů,

který se ovšem mění dle OS. Na řádce patnáct je potřeba ověřit, zda se paměť alokovala pomocí podmínky, který vypadá stejně jako u jiných programovacích jazyků. Uvnitř podmínky se volá metoda **strcpy_s**, která zkopíruje obsah proměnné `name` do proměnné `newName` s tím, že prostředním argumentem je velikost bufferu²², který v tomto případě je stejně velký jako alokovaná paměť pro proměnnou `newName`. Poté již připojí k tomu textovému řetězci čtyři nové znaky pomocí metody **strcat_s**, která má podobné argumenty jako metoda pro kopírování. Následně stačí již proměnnou vypsát, ale je nutné uvolnit alokovanou paměť, jelikož v tomto případě neexistuje mechanismus, který by paměť uvolnil. Uvolnění probíhá pomocí metody **free** s argumentem odkazu na alokovanou paměť.

Na řádce dvacet tři až dvacet osm je jednoduché porovnávání **if** a **else**. Na dvacátém devátém řádce je vidět inicializace pole stringů, které se inicializuje pomocí slova `char`, názvu proměnné a dvou hranatých závorek, kdy v prvních závorkách je počet stringů a ve druhých je jejich maximální velikost (znaků).

Pro iteraci skrz toto pole existuje **cyklus for**. Jediným rozdílem oproti ostatním jazykům je podmínka cyklu, kde se zjistí počet slov v dané proměnné. Toho je docíleno pomocí funkce **sizeof(words)**, která vrátí velikost alokované paměti a je vždycky násobkem maximální délky textových řetězců, v tomto případě deset. Kvůli tomu je nutné vydělit danou hodnotu maximální velikostí textových řetězců a tím vznikne číslo, které definuje počet prvků v poli. Uvnitř cyklu se pouze vypisuje prvek po prvku pomocí hrané závorek, které mají v sobě index daného prvku. Na řádcích třicet pět až třicet sedm je vidět obdobný cyklus, který na počáteční inicializaci využívá proměnnou `i`, definovanou výše, jelikož byla inicializována v rozsahu metody `main` a ne v cyklu.

Poslední ukázkou je tvorba metoda **addNumber**, jejíž hlavičku je definována právě na začátku programu. Poslední tři řádky definují kompletní metodu, která přijímá dvě čísla a vrací jejich součet. Syntaxe a použití této metody, viz řádek třicet devět, jsou velmi podobné většině programovacích jazyků. Větším rozdílem je absence klíčového slova **static**, které v jazyku C vůbec neexistuje, jelikož není nutné rozdělovat instanční a statické metody. To je dáno tím, že tento jazyk neobsahuje třídy.

²² Buffer – vyrovnávací paměť, do níž se ukládají data před jejich přesunem jinde

Zhodnocení

Programování v jazyce C je velmi složitou a zdlouhavou činností. Je zde potřeba pracovat manuálně s pamětí, což z tohoto jazyka dělá velmi obtížným pro výuku začátečníků. Pokud studenti nemají žádné předchozí znalosti programování, bude pro ně výuka velmi náročná. Nicméně se díky tomu naučí pracovat s pamětí, kterou za ně v jiných programovacích jazycích řeší garbage collector. Tím mohou získat cenné vědomosti, které je možno jinde opomenout, protože správa paměti probíhá automaticky. Ačkoliv se práce s knihovnami objevuje ve všech jazycích, zde je to absolutní nezbytností a je potřeba s tímto umět pracovat hned od začátku.

Z ukázkového kódu jasně vyplývá, že i jednoduché příkazy jsou dlouhé a zápis je tedy nejdelší ze všech jazyků. Navíc je zde potřeba dbát na velmi striktní syntaxi, jako je například psaní středníků za každým příkazem nebo dodržovat psaní velkých a malých písmen. Pro výuku na středních školách je tedy spíše nevhodným jazykem, pokud není cílem ve výuce probírat nízkoúrovňové koncepty.

6.1.2 C++

```
1. #include <iostream>
2.
3. using namespace std;
4.
5. int addNumber(int a, int b);
6.
7. int main()
8. {
9.     int number = 5;
10.    string name = "Test";
11.
12.    cout << number << endl << name << endl;
13.
14.    if (number >= 0) {
15.        cout << "Greater or equal to zero" << endl;
16.    }
17.    else {
18.        cout << "Less than zero" << endl;
19.    }
20.
21.    string array[] = {"Test", "Value"};
22.    for(string word : array) {
23.        cout << word << endl;
24.    }
25.
26.    for (unsigned int i = 0; i < 10; i++) {
27.        cout << "For index:" << i << endl;
28.    }
29.
30.    cout << addNumber(13, 67);
31.
32.    return 0;
33. }
34.
35. int addNumber(int a, int b) {
36.     return a + b;
37. }
```

Zdrojový kód v programovacím jazyce C++ začíná klíčovým slovem **#include**. Ten řekne preprocesoru, aby zahrnoval obsah, který se nachází v souborech, jenž je specifikován ve špičatých závorkách. V tomto případě jsou deklarovány objekty, které řídí čtení a zápis do standardních datových streamů, jako je vstup nebo výstup programu C++. Toto je převzato z jazyka C, jen jsou používány jiné knihovny.

Pro běžné objekty, jako je například výpis do konzole, je již definovaný **namespace std**, ten je efektivní uvést na začátku programu, viz řádek tři. Díky tomu je možné využívat jednotlivé příkazy jako je **cout** nebo **endl** bez specifikace tohoto namespace u každého použitého příkazu zvlášť (`std::cout`). Lze využívat nejen připravené namespace, ale i své vlastní pro zpřehlednění kódu.

I zde se zapisuje hlavička metody, které chceme využívat. Stejně jako u jazyka C se zde píše pouze hlavička včetně návratové hodnoty, názvu metody a jejích vstupních argumentů. Poté následuje metoda **main**, vstupní bod programu.

Na řádcích devět a deset lze vidět typickou deklaraci dvou proměnných. Protože je C++ staticky typované, je nutné vždy uvést datový typ, který se již poté nemůže změnit. Na řádku dvanáct je příklad výpisu do konzole pomocí klíčového slova **cout**, ten se používá ve spojení s operátorem `<<`, za nějž napíšeme proměnnou, kterou chceme vypsát. Funkce **endl** pak slouží k odřádkování.

Na řádku čtrnáct je možno vidět klasické využití podmínek pomocí klíčových slov **if** a **else**.

Pro vytvoření pole textových řetězců se v C++ využívá primitivní datový typ **string**, který nahrazuje pole charakterů z jazyka C. Není zde potřeba si přímo alokovat paměť, ani určovat, kolik pole zabere paměti, to dělá program sám. Do složených závorek se pak zapisují jednotlivé hodnoty.

Pro vypsání pole je zde použit **foreach** cyklus, který lze vidět na řádku dvacet dva. V hlavičce se nejdříve definuje datový typ, který se nachází v datové struktuře, která se specifikuje hned za dvojtečkou.

For cyklus se definuje stejně jako v jazyce C, na řádku dvacet šest je možno si všimnout jedné zvláštnosti při deklaraci řídicí proměnné cyklu. Klíčové slovo **unsigned** se využívá k tomu, aby proměnná nemohla nabývat negativních hodnot.

Na řádku třicet je volána metoda **addNumber**, která je definována pod hlavní metodou **main**. Zde je již dopsáno samotné tělo metody, sčítání dvou čísel, které jsou definovány jako vstupní argumenty metody. Hlavní metoda končí tím, že vrátí 0 a zároveň tím hlásí, že vše proběhlo v pořádku.

Zhodnocení

Ačkoliv je zápis příkazů v programovacím jazyce C++ podobným jako v jazyce C, lze si všimnout jistých detailů, které zápis zjednodušují. Například možnost zapisovat textový řetězec za pomoci již vytvořeného datového typu string, umožňuje studentovi jednodušší práci s textem obecně. Není tedy zapotřebí hned v prvních pár hodinách umět ovládat práci s polem pro to, aby mohl s textem pracovat. C++ navíc nabízí objektově orientované paradigma, které je v dnešní době velmi rozšířené a hojně využívané. Díky tomu se studenti naučí tyto základy, které lze pak využít v dalších programovacích jazycích. Pro studenty může být C++ určitou výzvou, kterou když překonají, ostatní programovací jazyky budou pro ně jednodušší se naučit. Pro výuku nízkourovňových je vhodným kandidátem.

6.1.3 C#

```
1. namespace ConsoleApp1;
2. public class Program
3. {
4.     static void Main(string[] args)
5.     {
6.         int number = 3;
7.         var text = "test";
8.         if (text == "test")
9.         {
10.            Console.WriteLine("is equal");
11.        }
12.        else
13.        {
14.            Console.WriteLine("not equal");
15.        }
16.        int[] array = { 2, 4, 5, 8, 4 };
17.        foreach(int number in array)
18.        {
19.            Console.WriteLine(number);
20.        }
21.        for (int i = 0; i <= 5; i++)
22.        {
23.            Console.WriteLine("for index is: " + i);
24.        }
25.        Console.WriteLine(AddNumber(12,number));
26.    }
27.    static int AddNumber(int a, int b)
28.    {
29.        return a + b;
30.    }
31. }
```

Zdrojový kód v jazyce C# začíná vždy klíčovým slovem **namespace**, což je přejato z jazyka C++. Ten označuje „jmenný prostor“ a slouží pro sdružování tříd pod jeden namespace. Využívá se především pro viditelnost tříd, pro to, aby spolu třídy spolupracovat, se musí vždy nacházet ve stejném namespace. Pokud se každá třída nachází v jiném namespace, je potřeba jí za pomoci klíčového slova **using** zviditelnit. Pro spuštění kódu je potřeba mít v programu vstupní bod – metodu Main, a vytvořenou třídu.

Při deklaraci proměnných je třeba určovat jejich datový typ, pokud se nejedná o proměnnou typu var. Na řádce sedm je typický příklad pro vytvoření proměnné pro celé číslo. Na řádce osm můžeme vidět vytvoření proměnné datového typu **var**, do které lze přiřadit jakoukoliv

hodnotu. Datový typ této proměnné si odvodí až samotný kompilátor, toto se nejčastěji využívá v případě, kdy se pracuje se složitými datovými typy, kdy by opakovaná explicitní deklarace vedla k nepřehlednému kódu.

Podmínka se též vytváří pomocí klíčového slova **if**, kdy do kulatých závorek je psán logický výraz. V tomto případě se porovnávají dva stringy a jejich hodnota pomocí **==**. I zde lze využít klíčové slovo **else** pro nesplnění podmínky. Pro výpis do konzole se v jazyce C# používá metoda **Console.WriteLine**, do níž se píše vstupní argumenty.

Pole se definuje pomocí hranatých závorek a musí mít specifikovaný datový typ. Je možné do něj ihned při inicializaci přidávat hodnoty, kterých má pole nabývat a podle nich se vytvoří velikost pole. Pokud se přidávají hodnoty při inicializaci, zapisují se do složených závorek a oddělují čárkou.

Pomocí klíčového slova **foreach** se vytvoří cyklus na procházení prvků v poli. V hlavičce cyklu je uveden datový typ proměnné, který se v poli nachází a definuje jakou datovou strukturu se prochází. V těle cyklu poté vypisují jednotlivé prvky.

For cyklus se vytváří stejně jako v jazyce C++. V hlavičce cyklu je deklarována řídicí proměnná, dále je uvedeno počet opakování a následně inkrementace proměnné. Do těla cyklu se opět píše sadu příkazů, které se mají opakovat.

Protože se jedná o jedinou třídu, další vytvořené metody, které se dají volat v metodě Main, musí být statické. Názvy metod se píše vždy s prvním velkým písmenem, což je velmi netradiční oproti ostatním programovacím jazykům. V hlavičce metody je zapotřebí definovat návratový datový typ, stejně jako se musí definovat datové typy vstupních argumentů. Tělo metody je obaleno složenými závkami, uvnitř se nachází kód s klíčovým slovem **return**, který provádí návrat z metody s konečným výsledkem.

Zhodnocení

Jazyk C# nabízí základní syntaxi, kterou lze najít ve většině jazyků. Kód je velmi přehledný, a pokud student dodrží pár předpisů jako je psaní středníků po každém příkazu nebo dodržování velkých a malých písmen, může být pro něj jednoduchým jazykem na naučení. Ovšem je zde zapotřebí počítat s tím, že student musí pochopit hned na začátku koncept OOP, protože bez toho nelze žádný kód napsat. Stejně tak jako používání jmenných prostorů.

Jinak jazyk nabízí velkou škálu externích knihoven a manuálů, které mohou studentům pomoci při tvorbě jejich projektů. Student se zde naučí syntaxi příkazů, které jsou velmi podobné i v jiných jazycích, a proto pro něj nebude problém přejít na jiný. C# lze zařadit do výuky hned začátečníkům díky jeho jednodušší syntaxi, než je C nebo C++.

6.1.4 Java

```
1. public class Main {
2.     public static void main(String[] args) {
3.         int number = 3;
4.         String text = "test";
5.
6.         if (text.equals("test")) {
7.             System.out.println("is equal");
8.         } else {
9.             System.out.println("not equal");
10.        }
11.
12.        int[] array = {2, 4, 5, 8, 4};
13.        for (int number : array) {
14.            System.out.println(number);
15.        }
16.
17.        for (int i = 0; i <= 5; i++) {
18.            System.out.println("for index is: " + i);
19.        }
20.
21.        System.out.println(addNumber(12, 7));
22.    }
23.
24.    public static int addNumber(int a, int b) {
25.        return a + b;
26.    }
27. }
```

Každý kód, který je psaný v jazyce Java, musí být uzavřen ve třídě, ve které se musí nacházet hlavní metoda **main**, která slouží jako vstupní bod ke spuštění programu. Metoda main může být napsána v jakékoliv třídě, není tedy zapotřebí, aby se nacházela přímo ve třídě Main. Metoda main ale musí být jen jedna a definována v konfiguraci spouštění.

Každá proměnná musí mít při své deklaraci určený datový typ. Proto hodnota, kterou je do ní uložena, musí mít stejný datový typ. **String** v Javě navíc není hodnotový datový typ, ale jedná se o takzvaný „referenční datový typ“, nebo též odvozený datový typ, přestože ho lze vytvořit stejným způsobem jako hodnotový typ. Ačkoliv ho lze vytvořit jako primitivní datový typ, ve skutečnosti se odkazuje na objekt.

Na řádku šest lze vidět jednoduché porovnání dvou textových řetězců. Porovnávání dvou proměnných v Javě je trochu složitější. Pokud se porovnávají primitivní datové typy, stačí

operátor `==` pro porovnání dvou hodnot, pokud se ale jedná o referenční datové typy, jako je právě například `String`, je potřeba použít jeho metodu `equals`, která umí porovnat dvě hodnoty. Pokud se `String` porovná přes operátor `==`, neporovnávají se hodnoty uvnitř proměnných, ale dva odkazy na řetězce ukazující na objekty v paměti.

Podmínky se v Javě píší stejně jako skoro v každém programovacím jazyce. Podmínka je uvedena klíčovým slovo `if`, za kterým následuje samotná podmínka. Dále lze využít i klíčové slovo `else`, který provede příkaz, pokud není podmínka splněna.

Na řádku dvanáct je představená jednoduchá deklarace pole. Stejně jako proměnná, i pole musí mít svůj daný datový typ, který definuje, jaké hodnoty lze do něj ukládat.

Pro cyklus `foreach` se používá pouze klíčové slovo `for`, kde se do hlavičky cyklu definuje datový typ proměnné a datová struktura, kterou se prochází. Na rozdíl od programovacího jazyka `C#` se zde tyto parametry oddělují dvojtečkou.

For cyklus, který je uveden na řádku čtrnáct, má obdobnou syntaxi jako je v dalších programovacích jazycích. Nejdříve je vytvořena „řídící proměnná“, dále je uvedena podmínka, která určuje, kolik iterací cyklus provede a následně je zapsána její inkrementace. Dvnitř cyklu se píší příkazy, které se mají opakovat.

Poslední částí příkladu syntaxe je využití metod. Pokud se metody vytváří ve třídě `Main`, tak musí být všechny statické, aby mohly být zavolány bez potřeby vytváření objektu. Metoda `addNumber` se nachází mimo metodu `main`, na dvacátém řádku. Do ní vstupují dva argumenty, které se následně v metodě sečtou a jejich součet metoda vrací zpátky. Metoda musí mít specifikovaný návratový datový typ, tudíž nelze vracet jiný, než který je zapsaný v hlavičce metody. Pokud se jedná o metodu s návratovým typem, metoda musí mít v sobě klíčové slovo `return`.

Metodu `addNumber` lze jednoduše zavolat v metodě `main`. Pro výpis součtu metody je zapotřebí zavolat metodu `System.out.println`, která umí své argumenty vypsát přímo do konzole.

Zhodnocení

Java je poměrně snadná na naučení a lze ji pochopit během krátké doby, protože její syntaxe je podobná angličtině, stejně jako `C#`. Pro studenty nabízí podobné možnosti, a kromě jiných

názvů metod a proměnných se jedná o podobně funkční jazyk. Java je case sensitivní a každý příkaz musí končit středníkem. Nabízí též OOP, jehož základy jsou potřeba znát hned na začátku, protože bez základní třídy nelze kód psát. Veškeré základy, které se studenti naučí lze jednoduše přenést do ostatních jazyků, a proto je vhodným pro výuku na střední škole.

6.1.5 JavaScript

```
1. let tmp = 5;
2. tmp = "Pět";
3.
4. const number = 10;
5. number = "Deset"; //TypeError: Assignment to constant
   variable.
6.
7. if(tmp >= 0) {
8.   console.log("Větší nebo rovno 0");
9. } else {
10.  console.log("Menší než 0");
11. }
12.
13. [25, "Test", true].forEach((value, index) => {
14.   console.log(`${value} at index ${index}`);
15. });
16.
17. for (let i = 0; i < 5; i++) {
18.   console.log(`For index is: ${i}`);
19. }
20.
21. function addNumber(a, b) {
22.   return a + b;
23. }
24. console.log(addNumber(13, 87));
```

Jelikož je JavaScript skriptovací jazyk, není zde potřeba ani třída nebo spustitelné metody, která by sloužily jako vstupní bod programu.

Během programu lze měnit datové typy jednotlivých proměnných. Proměnná **let** je základní proměnná, do které se ukládají jakékoliv hodnoty. Pro zamezení přepsání hodnoty je možno vytvořit proměnnou s klíčovým slovem **const** pro vytvoření neměnných referencí, viz řádek čtyři. Jakmile se v proměnné nachází nějaká hodnota, nelze již změnit a kód zahlásí error, který je lze vidět na řádku pět v komentáři.

Pro výpis do konzole se využívá **console.log**, statická metoda, která vypisuje hodnoty do konzole webového prohlížeče. Tato metoda se převážně využívá pro debuggování programu při vyhledávání chyb nebo ověřování funkčnosti programu. Místo klasických uvozovek pro definování stringu, lze použít gravis (``), který umožňuje vkládat proměnné přímo do textu. Toto použití je možno vidět na řádku osm, kde do textu je přidána proměnná *value*.

Na řádku třináct až patnáct je ukázán kód na tvorbu pole, které se však nekládá do žádné proměnné, ale pouze je využíváno pro jakousi logiku a dále s ním pracováno. Dalo by se říct, že se jedná o definici anonymního pole. Na toto pole je možno volat již předdefinovanou metodu **forEach**, která přijímá jako argument metodu, která se bude spouštět při každé iteraci. V tomto případě je vkládána do argumentu anonymní metoda (bezejmenná metoda), kterou je vytvořena pomocí syntaxe „arrow function“²³. Ta se definuje pomocí kulatých závorek, do kterých jsou psány argumenty funkce, a to hodnota iterovaného prvku a index pole. Následuje šipka a složené závorky, které indikuje tělo metody. Uvnitř se vypisuje každý prvek pomocí metody **console.log**.

Zajímavostí polí v JavaScriptu je ta skutečnost, že prvky v poli nemusí mít stejný datový typ. To je dáno tou skutečností, že pole není datovou strukturou, ale objektem, který může obsahovat hodnoty jakéhokoliv datového typu. (MDN contributors, 2023)

For cyklus se vytváří klasicky jako v každém programovacím jazyce. Nejdříve si je vytvořena řídicí proměnná, dále uvedena podmínka pro opakování a následně inkrementace proměnné.

Metody neboli funkce, jak se nazývají v JavaScriptu se deklarují pomocí klíčového slova **function** a její kód je zapsán ve složených závorkách. Vytvoření funkce **addNumber** lze vidět na řádce dvacet jedna. V hlavičce funkce není potřeba uvádět návratový typ, stačí využít klíčové slovo **return**, který definuje návratovou hodnotu. Není zde potřeba ani uvádět datový typ vstupních parametrů. Funkci je poté volána na řádce devatenáct, kde se vypisuje její hodnota do konzole.

Zhodnocení

JavaScript nabízí trochu jiný pohled do programování. Vzhledem k tomu, že je JS skriptovací jazyk umožňuje odlišný přístup v rámci psaní kódu. Není zde potřeba znát základy OOP, ačkoliv je možno je využít. Díky jeho dynamickému typování lze přistupovat k datům jinak a může se stát, že to pro studenty bude ze začátku matoucí. Nejsou zde takové restriktce, což umožňuje kombinovat různé věci přes sebe a díky tomu může dojít k nečekaným chybám. Nicméně jednoduché programy zde budou pro studenty lehké na

²³ Arrow function – neboli „šipková funkce“, zkrácený zápis pro vytvoření metody

programování a základní příkazy a funkce jsou zde obdobné jako v jiných programovacích jazycích. Je ovšem otázkou, zdali toto volnější programování bude pro studenty přínosem a nebude pro ně pak složité přecházet na striktnější jazyky. Pro studenty, kteří mají vyvíjet webové aplikace je vhodným jazykem, protože i zde se základy programování naučí.

6.1.6 PHP

```
1. <?php
2. $number = 10;
3. $number = "test";
4.
5. if ($number === "test") {
6.     echo "is equal\r\n";
7. } else {
8.     echo "is not equal\r\n";
9. }
10.
11. $array = ["Test", "Value"];
12. foreach ($array as $key => $value){
13.     echo "$value at index $key";
14. }

15. for ($i = 0; $i <= 5; $i++) {
16.     echo "For index is: $i\r\n";
17. }
18.
19. function addNumber($a, $b)
20. {
21.     return $a + $b;
22. }
23.
24. echo addNumber(12, 7);
```

Počátkem každého PHP souboru či scriptu je vždy řádek **<?php** (viz řádek jedna), který určuje, že uvnitř tohoto “tagu” lze psát PHP kód. Lze také uzavřít tento tag pomocí hesla **?>**, ale tato funkčnost se používá pouze u vkládání dat přímo do HTML soubory pomocí PHP.

Není zde přesně určen typ proměnných a jejich inicializace se dělá pomocí znaku **\$**, za kterým následuje název, jak lze vidět na řádku dva. Na dalším řádku lze tuto proměnnou nahradit jiným typem, respektive stringem.

Další sekci jsou podmínky, které jsou základem každého programovacího jazyka. Zde je rozdílné použití třech rovnítek, které oproti **==** udává, že se porovnává jak hodnota, tak i datový typ. Použití **else** je stejné jako v ostatních jazycích, respektive pokud se podmínka nesplní, provede se logika **else**. Uvnitř podmínky je použité klíčové slovo **echo**. Jedná se o takzvanou direktivu, která říká, jaká část se vypíše buď na obrazovku, nebo obecně vrátí do

bodů volání tohoto skriptu. V příkladu výše vypisuje string pomocí dvojitého uvozovky, které v PHP umožňuje vložení proměnné jednoduše pomocí napsání jejího názvu.

Největší předností jazyka PHP je práce s poli. Jejich inicializace je provedena pomocí hranatých závorek a následně libovolný počet proměnných stejného typu, v tomto případě string viz řádek jedenáct. Na dalším řádku lze jednoduše iterovat skrz tyto prvky pomocí cyklu **foreach**, které obsahuje v kulatých závorkách název iterované proměnné, klíčové slovo **as** a následně dočasné proměnné **\$key** a **\$value**, mezi kterými je šipka ($\Rightarrow$).

V proměnné **\$key** je index právě iterovaného prvku a v proměnné **\$value** je jeho hodnota. Pokud by byla vynechána proměnná **\$key** za výše zmíněnou šipku, fungoval by cyklus stejně a v proměnné **\$value** by byla stále hodnota iterovaného prvku. Uvnitř cyklu je standardní výpis pomocí metody **echo**.

Řádek jedenáct ukazuje syntaxi cyklů. Zde je využitý pouze **cyklus for**, který se skládá ze tří částí: inicializace, která deklaruje proměnnou, skrze kterou se cykluje, podmínka pro běh cyklu a poslední je dekrementace či inkrementace dané proměnné. Logikou uvnitř cyklu je pouze výpis indexu, který je v cyklu využíván.

Poslední a velmi podstatnou částí každého jazyka jsou funkce či metody. Ty jsou v PHP velmi jednoduché a přímočaré, jak je vidět na řádku patnáct. Podmínkou tvorby funkce je klíčové slovo **function**, za kterým následuje název metody a kulaté závorky pro její argumenty. V ukázce výše je metoda **addNumber**, do které vstupují dvě proměnné, které se sčítají. V těle metody je tedy pouze použito slovo **return**, které označuje koncový stav metody, respektive vrácení součtu dvou čísel pomocí operátoru plus. Posledním řádkem ukázky je výpis použití výše zmíněné metody.

Zhodnocení

PHP nabízí trochu odlišnou syntaxi oproti ostatním programovacím jazykům. Nejvíce odlišným je v rámci psaní proměnných a funkcí. Dle zápisu lze vidět, že je velmi krátký a spousta věcí lze napsat jednoduše. Jedná se též o skriptovací jazyk, takže není zapotřebí znát třídy ani metody hned na začátku. Jeho syntaxe není nijak složitá, je velmi podobná angličtině a příkazy mají logický smysl. Není zde potřeba se zabírat datovými typy, a to z něj činí jednoduchý jazyk pro naučení. Nevýhodou ovšem může být právě jeho volnost v zápisu dat, který se studenti nemusí nijak složitě učit a nejsou nuceni se tyto věci pochopit.

Stejně jako JavaScript je PHP vhodným programovacím jazykem pro výuku, pokud se mají studenti naučit vývoj webových aplikací.

6.1.7 Python

```
1. number = 10
2. number = 'test'
3.
4. if number == 'test':
5.     print('is equal')
6. else:
7.     print('is not equal')
8.
9. array = ["Test", "Value"]
10. for word in array:
11.     print(word)
12.
13. for i in range(10):
14.     print('For index is', i)
15.
16. def addNumber(a, b):
17.     return a + b
18.
19. print(addNumber(10, 5))
```

Python nevyžaduje pro psaní kódu žádnou třídu ani metodu. Není zde zapotřebí za každý příkazem psát středník, což umožňuje jednoduchost v psaní kódu a začínající programátor nemusí na něj pamatovat. Je to i určitá nevýhoda, protože většina programovacích jazyků toto vyžaduje. Zároveň je tu i velká absence závorek a se složenými závorkami se běžně u podmínek, funkcí nebo cyklů programátor nesetká.

Ani v Pythonu není potřeba u deklaraci proměnných psát datové typy. Proto je proměnná **number** na prvním řádku vytvořena jen za pomoci jejího názvu a přiřazení hodnoty do ní. Protože proměnné nemají definovaný datový typ, lze v jedné proměnné jednoduše střídát hodnoty, jako jsou čísla a textové řetězce. To je možno vidět na řádku dva.

Textový řetězec je zde vytvořen za pomoci jednoduchých, dvojitých nebo trojitých uvozovek. Díky tomuto se předchází problémům při práci s různými textovými formáty,

jako je například zapsání textu do konzole v uvozovkách. Není zde tedy potřeba využívat escapování²⁴.

Při vytváření podmínek v Pythonu platí stejné pravidlo a začíná se klíčovým slovem **if**. Podmínka začíná na řádce čtyři. Textové řetězce se zde porovnávají stejně jako v přechozích programovacích jazycích přes operátor `==`. Logický výraz podmínky je zakončen dvojtečkou, příkaz, který chceme, aby se provedl, pokud je podmínka splněna, musí být odsazený za pomoci tabulátoru. Díky tomu bude překladač vědět, že příkaz spadá pod podmínku. Podmínky i zde mají možnost klíčového slova **else**, který provede příkazy, pokud podmínka není splněna.

Python má určitě nejjednodušší metodu pro vypsání textu nebo hodnot do konzole. Pro výpis se používá metoda **print**, která je velmi jednoduchá na zapamatování.

Co se výrazně liší od ostatních programovacích jazyků je **for cyklus**. I zde je využita řídicí proměnná cyklu, ovšem není zapotřebí ji složitě deklarovat, jak je možno vidět na řádce devět. Místo toho je zde využita funkce **range**, která vrací vygenerovaná čísla v rozsahu, který je definován vstupním argumentem. Celá hlavička for cyklu je pak zakončena stejně jako u podmínky dvojtečkou.

I zde není zapotřebí vytvářet složité metody, protože se metoda nenachází uvnitř třídy, která by spouštěla kód. Metody se vytvářejí pomocí klíčového slova **def**, za kterým následuje pouze název. Do kulatých závorek se definují vstupní argumenty, které též nepotřebují upřesnění datového typu. Klíčovým slovem **return** se vrací hodnoty z metod, jak je možno vidět na řádce třináct, kde se vrací součet dvou čísel.

Zavolání metody je stejné jako u dalších programovacích jazyků pro vypsání výsledku je zapotřebí použití metody **print**.

Zhodnocení

Díky příkladu lze jasně vidět, že jeho zápis základní příkazů je nejkratší. Není zde nutnost psát středníky ani datové typy nebo složené závorky funkcí. Je zde pak na učiteli, zdali tyto prvky probere, aby měli studenti alespoň základní vhled do této problematiky. Pro začínající

²⁴ Escapování – před speciální znak vkládáme zpětné lomítko (`\`), který udává, aby následující znak byl interpretován doslova jako znak, nikoliv jako jeho speciální význam

programátory může být zápis metod nebo podmínek velmi matoucí, protože na první pohled nevidí, kde začínají, a kde končí. Jeho syntaxe je velmi odlišná od ostatních programovacích jazyků a jeho volnost v zápisu nemusí být vždy užitečná. Vzhledem k tomu, že je jeho syntaxe velmi podobná angličtině, může tato odlišnost přispět k rychlejšímu naučení psaní kódu. Tato jednoduchost přispívá k lepšímu čtení kódu pro začátečníky a díky tomu lze označit Python jako správný výběr pro výuku vývoje aplikací.

6.1.8 TypeScript

```
1. let tmp = 5;
2. tmp = "Pět";      //TS2322: Type string is not assignable
to type number
3.
4. const number = 10;
5. number = "Deset"; //Attempt to assign to const or readonly
variable
6.
7. if (tmp >= 0) {
8.     console.log("Greater or equal than 0");
9. } else {
10.    console.log("Less than 0");
11. }
12.
13. const arr = [25, "Test", true] as const;
14.
15. [25, "Test", true].forEach((value, index) => {
16.    console.log(`${value} at index ${index}`);
17. });
18.
19. for (let i = 0; i < 5; i++) {
20.    console.log(`For index is: ${i}`);
21. }
22.
23. function addNumber(a: number, b: number): number {
24.    return a + b;
25. }
26.
27. console.log(addNumber(13, 87));
```

TypeScript vychází ze skriptovacího jazyka JavaScript a v rámci syntaxe toho mají hodně společného. TypeScript je například obohacený o statické typování, třídy a rozhraní. Oproti JavaScript dokáže upozornit již při vyvíjení kódu a program nedovolí ani zkompileovat, pokud se v něm objevuje nějaká chyba. Není zde potřeba žádný vstupní bod, ani třída nebo metoda.

Na řádku jedna je vytvořena stejná proměnná jako v JS. Rozdíl mezi dvěma jazyky lze vidět na řádku dva, jakmile totiž v TS je přiřazena do proměnné hodnota, automaticky se proměnná převede do jejího datového typu, který již nelze změnit. Při kompilaci kódu by druhý řádek vyhodil chybu, protože nelze přiřazovat do číselné proměnné text. Co se ale

nemění je klíčové slovo **const** pro proměnné. I v tomto případě je proměnná neměnná, a pokud je do ní uložena hodnota 10, musí už takto zůstat.

Podmínky jsou v TypeScriptu stejné jako v dalších programovacích jazycích. Využívají se zde klíčová slova **if** a **else**. TypeScript využívá stejnou metodu pro výpis do konzole **console.log**, stejně jako zápis stringu včetně gravisu.

Na řádku patnáct lze vidět deklaraci konstantního pole s konstantními hodnotami. Klíčová slova **as const** definují, že toto pole je neměnné. Toto vytvoření pole je oproti JavaScriptu jiné, nic takového tam totiž použít nelze.

Foreach v TypeScriptu je stejný jako u jazyka JavaScript. Využití **for** cyklu je též stejné jako ve většině programovacích jazycích. Je zde využita řídicí proměnná, která je typu **let**, datový typ se tudíž přiřadí až za běhu programu.

Největší změna v zapisování kódu se týká jednotlivých funkcí. Ty se označují pomocí klíčového slova **function**, ale v hlavičce se definuje nejen návratový typ, ale i datové typy vstupních argumentů, jak je možno vidět na řádce dvacet čtyři. Těm se nejdříve dá identifikátor a následně za pomoci dvojtečky přidá datový typ. Návratový typ se netradičně značí až za vstupními argumenty pomocí dvojtečky. Funkce klasicky končí pomocí klíčového slova **return**.

Funkci je volána poté na řádce dvacet osm, stejně jako u ostatních jazyků ve funkci, která vypíše její hodnotu do konzole.

Zhodnocení

TypeScript nabízí velmi obdobné možnosti psaní kódu jako JavaScript až na jednu výjimku, jeho statické typování nutí studenta dodržovat striktnější podmínky a více sledovat to, co píše. Ani zde není potřeba znát třídy ani metody a jednoduše student může začít psát kód. Oproti JS navíc zahlásí chybu už při kompilaci kódu a tím zjednodušuje kontrolu chyb, a to pro studenty může být užitečné. Může tedy sloužit jako obdoba právě JavaScriptu, pokud se chce učitel více soustředit na datové typy a webové aplikace.

6.2 Možnosti využití

Důležitým aspektem při výběru jazyka je i vhodnost pro psaní konkrétních aplikací. Některé programovací jazyky jsou obecně využívány napříč různými obory a jiné jsou čistě specializované pro konkrétní oblasti, proto je velmi důležité znát jejich přednosti.

6.2.1 Desktopové aplikace

Při rozmyšlení, jaké programovací jazyky lze využít, by mělo být bráno v úvaze pár základní věci: koncept celé aplikace, její vzhled, operační systém a funkce každého jazyka.

Programovací jazyk C# je multiparadigmový jazyk, nabízí tedy možnost jeho využití jakýmkoliv způsobem. Díky .NET Frameworku, který umožňuje komplexní vývoj, pokročilou úroveň programování a efektivitu při návrhu, vývoji, testování a nasazení programu, lze velmi příjemným způsobem vytvářet aplikace. (Gadhavi, 2023) Nabízí bohatou sadu funkcí pro vývoj grafických uživatelských rozhraní, zpracování vstupních a výstupních operací a přístup k systémovým prostředkům. Navíc podporuje rychlý vývoj aplikací díky rozsáhlým nástrojům a knihovnám. Aplikace v jazyce C# jsou však primárně omezeny na platformu Windows, což může být pro některé projekty omezením. (Yaseen, 2023)

V programovacím jazyce Java mohou vývojáři snadno vytvářet desktopové aplikace pomocí nástrojů jako je rozhraní API – Swing, AWT nebo JavaFX. Java navíc nabízí příjemné využití OOP programování a je multiplatformní, což znamená, že naprogramovaný software lze spustit v jakémkoliv operačním systému s JVM. (Kayasth, 2022)

I v programovacím jazyce Python lze najít spousta knihoven a modulů, které je možno využít pro GUI při vyvíjení desktopových aplikací, navíc psaní kódu je minimální, což dokáže zrychlit proces vývoje. (Yaseen, 2023)

Ačkoliv lze v programovacím jazyce C vytvářet aplikační software, operační systémy nebo ovladače zařízení, dnes se běžně používá pro software mikrokontrolerů nebo vestavěných systémů²⁵. Programovací jazyk C je doporučeným jazykem pro vytváření ovladačů a aplikací vestavěných systémů. Dostupnost hardwarových rozhraní API na strojové úrovni,

²⁵ Vestavěný systém – jednoúčelový počítač

stejně jako přítomnost překladačů jazyka C, dynamické přidělování paměti nebo deterministická spotřeba zdrojů činí tento jazyk nejoblíbenějším. (Ravikiran, 2024)

6.2.2 Webové aplikace

JavaScript je přizpůsobivý programovací jazyk, který lze využít jak u front-end, tak i back-end vývoje. Na straně klienta je možno také využít Angular a na straně serveru Node.js. Nabízí i další knihovny pro vývoj jako je React a Electron. Znalost JavaScriptu je v podstatě při vývoji front-endu nevyhnutelná, protože je klíčový při vytváření a správě dynamického obsahu na webových stránkách. (Yadav, 2023b)

TypeScript nabízí v této oblasti velmi podobné využití jako JavaScript akorát v jiné syntaxi.

PHP je nejzákladnější skriptovací jazyk na straně serveru, který je integrován do jazyka HTML. Vývojářům tak poskytuje interakci s databází a umožňuje vytvářet dynamický obsah stránek. PHP nabízí širokou škálu vestavěných funkcí pro úlohy, jako je manipulace s databází, práce se soubory a šifrování dat. Navíc většina poskytovatelů webhostingu nabízí výchozí podporu jazyka PHP, což zjednodušuje proces nasazení aplikací PHP. (Yadav, 2023b)

Python obsahuje mnoho výkonných frameworků pro vývoj webových aplikací, například Flask a Django. Oba frameworky jsou extrémně rychlé, bezpečné a škálovatelné. Díky těmto možnostem mohou vytvářet jednoduché webové aplikace i začátečníci. (Yadav, 2023b)

6.2.3 Mobilní aplikace

Pokud se chceme zaměřit na vývoj aplikací pro prostředí iOS, jednoznačně vyhrávají programovací jazyky Swift a Objective-C. Programovací jazyk Swift je momentálně velmi populární a nabízí SwiftUI pro vývoj uživatelského rozhraní. Oproti Objective-C má přirozenější a čitelnější syntaxi. (Parmar, 2023)

Objective-C je programovací jazyk C s nadstavbou pro objektové programování. Má složitější syntaxi, nižší úroveň zabezpečení a menší funkcionalitu než Swift, proto se od něj pomalu upouští. (Sagar, 2023)

Pro vývoj Android aplikací je nejčastěji používána Java nebo Kotlin, ovšem někteří používají i programovací jazyk Python. Rostoucí požadavky na zabezpečení aplikací, nižší náklady na vývoj, tvorba aplikací na vysoké úrovni, úspora zdrojů díky vývoji napříč

platformami, vysoká škálovatelnost a všestrannost, pokročilé testovací funkce a udržování aktuálních trendů jsou některé z hlavních důvodů, proč vývojáři upřednostňují Javu před alternativními technologiemi. (Koteshov, 2023)

Programovací jazyk Kotlin využívá JVM. Má dlouhý seznam funkcí a knihoven, které se hodí pro vytváření aplikací. Nabízí trochu jednodušší a stručnější syntaxi, která zlepšuje implementaci složité logiky. (Makhija, 2023)

6.2.4 Datová analýza a umělá inteligence

Jednoznačně nejpobulárnější programovací jazyk v této oblasti je Python, a to díky své jednoduchosti a flexibilitě. Umožňuje práci s daty ve všech odvětvích, díky velkému množství knihoven, které nabízí. V datové vědě se běžně používá pro zpracování dat, implementaci algoritmů datové analýzy a trénování algoritmů strojového učení a hlubokého učení. (Luna, 2023) Skutečná síla jazyka Python v oblasti umělé inteligence pramení z jeho rozsáhlých knihoven a frameworků, jako jsou TensorFlow, PyTorch a Keras, které nabízejí předpřipravené funkce a třídy, jež výrazně urychlují vývoj složitých algoritmů. (Gupta, 2024)

Java je také velmi vhodným kandidátem pro projekty umělé inteligence, zejména pro rozsáhlé aplikace na podnikové úrovni. Její platformová nezávislost je velkou výhodou, protože umožňuje snadný přenos systémů AI vyvinutých v jazyce Java do různých prostředí bez nutnosti rekompile. Jednou z pozoruhodných předností Javy v oblasti umělé inteligence je rostoucí oblast zpracování přirozeného jazyka a neuronových sítí. Programovací knihovny jako Deeplearning4j nabízejí robustní nástroje pro vytváření složitých struktur neuronových sítí. Silná správa paměti a vysoce výkonné mechanismy pro garbage collection v jazyce Java zajišťují efektivní práci s velkými soubory dat typickými pro umělou inteligenci. (Gupta, 2024)

Jazyk C++ je známý svou vysokou rychlostí provádění a kontrolou nad systémovými prostředky, takže je ideální pro vývoj aplikací umělé inteligence, kde je výkon kritickou součástí. Jeho schopnost běžet v blízkosti systémového hardwaru umožňuje efektivnější zpracování algoritmů umělé inteligence, což je zásadní v aplikacích reálného času a systémech s omezenými zdroji. Také se běžně používá při vývoji her a simulací, což jsou

oblasti, které stále více využívají umělou inteligenci pro realističtější a složitější prostředí.
(Gupta, 2024)

Závěr

Cílem práce bylo představit a porovnat programovací jazyky, které by mohly být vhodné pro výuku programování na středních školách. Práce obsahuje rozdělení programovacích jazyků podle programovacích paradigmat a jejich historii, vysvětlení základních pojmů a specifikace každého z nich.

Navržený výběr programovacích jazyků vychází nejen z hlediska oblíbenosti mezi vývojáři a pracovními nabídkami, ale aby programovací jazyk naplňoval potřeby v rámci RVP SOV.

Nejdůležitější částí této práce bylo porovnání jednotlivých jazyků z různých perspektiv. První aspekt bylo zapsání programu dle syntaxe každého jazyka. Jsou zde připraveny základní příkazy pro každý jazyk, které jsou poté vysvětleny a zhodnoceny. Díky tomu jsou patrné na první pohled rozdíly mezi nimi, a to nejen v samotném zápisu kódu, ale také i v rámci funkcionality jazyka. Je nutné podotknout, že nelze s přesností říci, která syntaxe jazyka je nejsložitější nebo naopak nejjednodušší, protože se jedná o velmi subjektivní aspekt. Ovšem díky těmto příkladům je možné pozorovat a vizualizovat si základní řešení problémů. Dále je možné poukázat i některé problémy, se kterými se lze setkat během vývoje v daném programovacím jazyce.

Následně jsou jazyky porovnány a doporučeny v druhém aspektu, z hlediska jejich využití již v konkrétní oblasti IT. Je důležité si uvědomit jejich přednosti a ty využít přesně v tom správném směru. Proto jsou u každé oblasti navrženy nejpoužívanějších jazyky, které by mohly sloužit jako základní kámen pro rozhodnutí při výběru.

Na konec je důležité zmínit, že nelze označit jeden jazyk jako vyhovující pro všechny. Většina programovacích jazyků nabízí základní principy, které je potřeba se během výuky naučit a lze je tedy označit za vhodné. Při rozhodování, který jazyk použít, je nutné se zamyslet nad tím, jaký typ aplikací se mají studenti naučit vyvíjet. Podle typu aplikace je možné uvažovat nad konkrétními jazyky a vybrat si přesně ten, který splňuje požadavky ŠVP. Jakmile má učitel představu o průběhu jeho hodin, je jednoduché si vybrat jazyk z výše uvedených kategorií. Při výběru je také potřeba započítat nezastavitelný vývoj technologií. Často se stává, že jazyky nebo jejich některé prvky byly důležité při vývoji aplikací dnes, ale nebudou již relevantní za několik let.

Seznam použitých informačních zdrojů

AGARWAL, Chirag a MIGLANI, Gaurav. *Garbage Collection in Java*. Online. GeeksforGeeks. 2022. Dostupné z: <https://www.geeksforgeeks.org/garbage-collection-java/>. [cit. 2023-11-15].

ASHTARI, Hossein. *Machine Language vs. Assembly Language: Key Differences*. Online. Spiceworks. 2023, 18.4.2023. Dostupné z: <https://www.spiceworks.com/tech/tech-general/articles/machine-vs-assembly-language/>. [cit. 2023-11-05].

BAELDUNG. *Compiled vs. Interpreted Programming Languages*. Online. Baeldung on Computer Science. 2023. Dostupné z: <https://www.baeldung.com/cs/compiled-vs-interpreted-languages>. [cit. 2023-11-12].

BAELDUNG. *Statically Typed vs Dynamically Typed Languages*. Online. Baeldung on Computer Science. 2023. Dostupné z: <https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages>. [cit. 2024-07-05].

BASHARAT, Sehrish. *Difference between procedural and non-procedural languages*. Online. Edu input. 2022, 20.10.2022. Dostupné z: <https://eduinput.com/difference-between-procedural-and-non-procedural-languages/>. [cit. 2023-11-05].

BASUMALLICK, Chiradeep. *What Is OOP (Object Oriented Programming)? Meaning, Concepts, and Benefits*. Online. Spiceworks. 2022. Dostupné z: <https://www.spiceworks.com/tech/devops/articles/object-oriented-programming/>. [cit. 2023-11-12].

BHARADWAJ, Anjan. *Scripting vs. Programming Languages: What's the Difference?* Online. Indeed.com. 2023. Dostupné z: <https://www.indeed.com/career-advice/career-development/scripting-vs-programming>. [cit. 2023-11-27].

BYJU'S EXAM PREP. *Difference Between Compiled and Interpreted Language*. Online. BYJU'S Exam prep. Uttar Pradesh: BYJU'S Exam prep, November 11th, 2022. Dostupné z: <https://byjusexamprep.com/difference-between-compiled-and-interpreted-language-i>. [cit. 2023-04-09].

CONTRIBUTORS, Wikipedia. *Python syntax and semantics*. Online. In: Wikipedia, The Free Encyclopedia. 2024. Dostupné z: https://en.wikipedia.org/wiki/Python_syntax_and_semantics. [cit. 2024-07-06].

COURSERA. *What Is C ? (And How to Learn It)*. Online. Coursera. 2023. Dostupné z: <https://www.coursera.org/articles/what-is-c-plus-plus>. [cit. 2023-11-13].

COURSERA. *What Is Java Used For?* Online. Coursera. 2023. Dostupné z: <https://www.coursera.org/articles/what-is-java-used-for>. [cit. 2023-11-15].

COURSERA. *What Is Python Used For? A Beginner's Guide*. Online. Coursera. 2023. Dostupné z: <https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python>. [cit. 2023-11-15].

COURSERA. *5 Types of Programming Languages*. Online. Coursera. 2023, 12.10.2023. Dostupné z: <https://www.coursera.org/articles/types-programming-language>. [cit. 2023-11-05].

DAYLIGHT, Edgar G. Towards a Historical Notion of 'Turing—the Father of Computer Science'. Online. *History and Philosophy of Logic*. 2015, roč. 36, č. 3, s. 205-228. ISSN 0144-5340. Dostupné z: <https://doi.org/10.1080/01445340.2015.1082050>. [cit. 2023-06-18].

DENEIRE, Tom. *What every programmer should know about Alan Turing*. Online. Medium. Medium, 2021, 15.11.2021. Dostupné z: <https://tomdeneire.medium.com/what-every-programmer-should-know-about-alan-turing-70e8d73be77c>. [cit. 2023-06-18].

DOHERTY, Erin. *What is object-oriented programming? OOP explained in depth*. Online. Educative. Educative, 2020. Dostupné z: <https://www.educative.io/blog/object-oriented-programming>. [cit. 2023-04-23].

GADHAVI, Maitray. *The Legacy of .NET Frameworks for Desktop Application Development*. Online. Radixweb. 2023. Dostupné z: <https://radixweb.com/blog/net-preferred-desktop-application-development-technology#Choose>. [cit. 2023-11-28].

GUPTA, Sakshi. *The Top 5 AI Programming Languages You Need as an AI Engineer*. Online. Springboard Blog. 2024. Dostupné z: <https://www.springboard.com/blog/data-science/best-programming-language-for-ai/>. [cit. 2024-07-08].

HELLER, Martin. *What is Kotlin? The Java alternative explained*. Online. InfoWorld. 2022. Dostupné z: <https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html>. [cit. 2023-11-15].

HEMMENDINGER, David. *ALGOL*. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/ALGOL-computer-language>. [cit. 2023-11-25].

HEMMENDINGER, David. *Pascal*. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/Pascal-computer-language>. [cit. 2023-11-25].

HERRITY, Kyle. *What Is Object-Oriented Programming (OOP)? A Complete Guide*. Online. Indeed.com. 2023. Dostupné z: <https://www.indeed.com/career-advice/career-development/what-is-object-oriented-programming>. [cit. 2023-11-12].

HIERSO, Severine. *Coding From 1849 to 2022: a Guide to The Timeline of Programming Languages*. Online. IEEE COMPUTER SOCIETY. Washington, D.C., USA: IEEE, 2022, 6.10.2022. Dostupné z: <https://www.computer.org/publications/tech-news/insider-membership-news/timeline-of-programming-languages>. [cit. 2023-04-17].

HISTORY COMPUTER STAFF. *Plankalkül Programming Language: History, Origin, and More*. Online. History-Computer. 2023. Dostupné z: <https://history-computer.com/plankalkul-guide/>. [cit. 2023-11-18].

CHRISTOPHER, Esther. *How Does JavaScript Work Behind the Scenes? JS Engine and Runtime Explained*. Online. FreeCodeCamp.org. 2023. Dostupné z: <https://www.freecodecamp.org/news/how-javascript-works-behind-the-scenes/>. [cit. 2023-11-15].

INDIANA UNIVERSITY. *What is the difference between a compiled and an interpreted program?* Online. University Information Technology Services. Indiana: University Information Technology Services, 2021. Dostupné z: <https://kb.iu.edu/d/agsz>. [cit. 2023-04-09].

ISAAC COMPUTER SCIENCE. *High- and low-level languages*. Online. Isaac Computer Science. Isaac Computer Science, 2019. Dostupné z: https://isaaccomputerscience.org/topics/programming_languages?examBoard=all&stage=all. [cit. 2023-04-16].

- JAIN, Mayank. *How is a C Program Compiled and Executed?* Online. Scaler Topics. 2022, 4.5. 2023. Dostupné z: <https://www.scaler.com/topics/how-to-compile-cpp/>. [cit. 2023-11-15].
- JALLI, Artturi. *What Is C#?* Online. Built In. 2022. Dostupné z: <https://builtin.com/software-engineering-perspectives/c-sharp>. [cit. 2023-11-15].
- KANTOR, Ilya. *An introduction to JavaScript*. Online. JavaScript. 2007. Dostupné z: <https://javascript.info/intro>. [cit. 2023-11-15].
- KAPIAS, Michal. *Top Most Popular Programming Languages for Mobile App Development*. Online. Fireart Studio. 2023. Dostupné z: <https://fireart.studio/blog/top-most-popular-programming-languages-for-mobile-app-development/>. [cit. 2023-11-28].
- KARANI, Dhruvil. *How does Python work?* Online. Medium. 2020. Dostupné z: <https://towardsdatascience.com/how-does-python-work-6f21fd197888>. [cit. 2023-11-15].
- KAYASTH, Dhruv. *What Is The Best Java Framework For Desktop Applications?* Online. Infiraise. 2022. Dostupné z: <https://www.infiraise.com/what-is-the-best-java-framework-for-desktop-applications/>. [cit. 2023-11-28].
- KERNIGHAN, Brian W. a RITCHIE, Dennis M. *Programovací jazyk C*. 2. vydání. Přeložil Zbyněk ŠTÁVA. Brno: Computer Press, 2019. ISBN 978-80-251-4965-2.
- KHAN, Ahmed. *Getting Started with PHP Programming*. Online. The Official Cloudways Blog. 2021. Dostupné z: <https://www.cloudways.com/blog/how-to-start-with-php/>. [cit. 2023-11-26].
- KIDDER, T.J. *A Brief History of Programming Languages: Part I*. Online. LEARN academy. 2023. Dostupné z: <https://learnacademy.org/blog/first-programming-language-use-microsoft-apple/>. [cit. 2023-11-25].
- KING, K. N. *C Programming: A Modern Approach*. 2008. W. W. Norton & Company, 2008. ISBN 978-0-393-97950-3.
- KNUTH, Donald Ervin. *Umění programování*. Brno: Computer Press, 2008. ISBN 978-80-251-2025-5.

KNUTH, Donald Ervin. *Umění programování*. Brno: Computer Press, 2010. ISBN 978-80-251-2898-5.

KOTESHOV, Dmitri. *Java in Mobile Application Development: Reasons and Benefits of Using in 2023 and Beyond*. Online. EPAM Startups. 2023. Dostupné z: <https://anywhere.epam.com/business/java-for-mobile-app-development>. [cit. 2023-11-28].

KRÁLOVÁ, Mgr. Magda. *ADA LOVELACE*. Online. Edu Techmania. Plzeň: Techmania Science Center, 2007. Dostupné z: <http://edu.techmania.cz/cs/encyklopedie/vedec/1243/lovelace>. [cit. 2023-04-17].

LARSON, Quincy. *Just in Time Compilation Explained*. Online. FreeCodeCamp. FreeCodeCamp, 2020. Dostupné z: <https://www.freecodecamp.org/news/just-in-time-compilation-explained/>. [cit. 2023-04-09].

LESTAL, Justin. *History of programming languages*. Online. DevSkiller - Powerful tool to test developers skills. 2023. Dostupné z: <https://devskiller.com/blog/history-of-programming-languages/>. [cit. 2023-11-25].

LUNA, Javier Canales. *Top programming languages for data scientists in 2023*. Online. DataCamp. 2023. Dostupné z: <https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022>. [cit. 2023-11-28].

LUZMO TEAM. *Using AI for Data Analysis: The Ultimate Guide (2024)*. Online. Nenalezený vydavatel. 2023. Dostupné z: <https://www.luzmo.com/blog/ai-data-analysis>. [cit. 2024-07-08].

MAKHIJA, Ravi. *12 Reasons for Using Kotlin for Android App Development*. Online. Stackify. 2023. Dostupné z: <https://stackify.com/12-reasons-for-using-kotlin-for-android-app-development/>. [cit. 2023-11-28].

MATOUŠEK, David. *C++ bez předchozích znalostí*. Brno: Computer press, 2016. ISBN 978-80-251-4640-8.

MCCRACKEN, Harry. *Fifty Years of BASIC, the Programming Language That Made Computers Personal*. Online. Time. 2014. Dostupné z: <https://time.com/69316/basic/>. [cit. 2023-11-25].

- MDN CONTRIBUTORS. *Array*. Online. MDN Web Docs. 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array. [cit. 2023-11-26].
- MŠMT ČR. *Rámcový vzdělávací program pro obor vzdělání 18 – 20 – M/01 Informační technologie*. Online. Edu.cz. 2023. Dostupné z: <https://www.edu.cz/rvpsov/ciste/18-20-M01.pdf>. [cit. 2023-11-25].
- MŠMT ČR & NPI ČR. *Revize ICT*. Online. Revize-ict-sov.rvp.cz. 2023. Dostupné z: <https://revize-ict-sov.rvp.cz>. [cit. 2023-11-25].
- MUNRO, André. *Python*. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/Python-computer-language>. [cit. 2023-11-15].
- MURAT. *Decoding the Debate: Interpreted vs. Compiled Programming Languages*. Online. Medium. 2023. Dostupné z: <https://muratakan.medium.com/decoding-the-debate-interpreted-vs-compiled-programming-languages-b5551c2f0770>. [cit. 2023-11-12].
- NEHA, T. *Difference Between Procedural and Non-procedural Language*. Online. Tech Differences. 2023. Dostupné z: <https://techdifferences.com/difference-between-procedural-and-non-procedural-language.html>. [cit. 2023-11-12].
- NEZNÁMÝ. *What is the difference between a compiled and an interpreted program?: What is the difference between a compiled and an interpreted program?* Online. University Information Technology Services. 2023. Dostupné z: <https://kb.iu.edu/d/agsz>. [cit. 2023-11-12].
- NIKOV, Dim. *What Is a Computer Program?* Online. Makersaid. 2022, 21.8.2022. Dostupné z: <https://makersaid.com/what-is-a-computer-program/>. [cit. 2023-11-05].
- NISHAD, Ankit. *Classes and Objects in Object Oriented Programming*. Online. Enjoyalgorithms. 2023. Dostupné z: <https://www.enjoyalgorithms.com/blog/classe-and-object-in-oops>. [cit. 2023-11-12].
- NOLLE, Tom. *Structured programming (modular programming)*. Online. Techtarger. USA: Techtarger, 2023. Dostupné z:

<https://www.techtarget.com/searchsoftwarequality/definition/structured-programming-modular-programming>. [cit. 2023-06-17].

NPI ČR. *Co jsou rámcové a školní vzdělávací programy (RVP a ŠVP)*. Online. Infoabsolvent.cz. 2023. Dostupné z: <https://www.infoabsolvent.cz/Rady/Clanek/7-0-13>. [cit. 2023-11-25].

OLUMIDE, Shittu. *What is Java Used For in 2023? The Java Programming Language and Java Platform Strengths*. Online. FreeCodeCamp.org. 2023. Dostupné z: <https://www.freecodecamp.org/news/what-is-java-used-for/>. [cit. 2023-11-15].

ORACLE. *The Java Language Environment*. Online. Oracle. 2023, 2023. Dostupné z: <https://www.oracle.com/java/technologies/introduction-to-java.html>. [cit. 2023-11-15].

ORACLE. *About the Java Technology*. Online. Oracle. 1995, 2022. Dostupné z: <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>. [cit. 2023-11-15].

PANIGRAHI, Kiran Kumar. *Difference Between Procedural and Non-procedural Language*. Online. Tutorialspoint. 2022, 7.12.2022. Dostupné z: <https://www.tutorialspoint.com/difference-between-procedural-and-non-procedural-language>. [cit. 2023-11-05].

PANIGRAHI, Kiran Kumar. *Difference Between High-Level Language and Low-Level Language*. Online. Tutorialspoint. Tutorialspoint, 2023. Dostupné z: <https://www.tutorialspoint.com/difference-between-high-level-language-and-low-level-language>. [cit. 2023-04-16].

PANIGRAHI, Kiran Kumar. *Difference Between Procedural and Non-procedural Language*. Online. Tutorialspoint. 2022. Dostupné z: <https://www.tutorialspoint.com/difference-between-procedural-and-non-procedural-language>. [cit. 2023-11-12].

PARMAR, Ashish. *Swift for iOS Development: 8 Reasons determining Why It is Brilliant*. Online. Prismetric. 2023. Dostupné z: <https://www.prismetric.com/using-swift-for-ios-app-development/>. [cit. 2023-11-28].

- PARUCH, Zach. *What Is JavaScript*. Online. Semrush Blog. 2023. Dostupné z: <https://www.semrush.com/blog/javascript/>. [cit. 2023-11-15].
- PATADIYA, Jaydeep. *A Beginner's Guide to Desktop App Development*. Online. Radixweb. 2023. Dostupné z: </blog/desktop-application-development-guide>. [cit. 2023-11-28].
- PATEL, Sandip a PAREKH, Dhruti. *How does PHP work internally?* Online. Learn2torials. 2021. Dostupné z: <https://learn2torials.com/a/how-php-script-works>. [cit. 2023-11-16].
- PEDAMKAR, Priya. *What is Groovy?* Online. EDUCBA. 2023. Dostupné z: <https://www.educba.com/what-is-groovy/>. [cit. 2023-11-15].
- PREPBYTES. *What is Interpreted Language?* Online. PrepBytes Blog. 2023. Dostupné z: <https://www.prepbytes.com/blog/general/what-is-interpreted-language/>. [cit. 2023-11-12].
- PRIYA, Mohana. *High and Low Level Languages*. Online. TEACH COMPUTER SCIENCE. TEACH COMPUTER SCIENCE, 2023. Dostupné z: <https://teachcomputerscience.com/high-and-low-level-languages/>. [cit. 2023-04-16].
- RAVIKIRAN, A. S. *Use of C Language: Everything You Need to Know*. Online. Simplilearn.com. 2023. Dostupné z: <https://www.simplilearn.com/tutorials/c-tutorial/use-of-c-language>. [cit. 2023-11-13].
- RAVIKIRAN, A. S. *Getting Started with C Programming: A Beginner's Guide!*. Online. Simplilearn.com. 2024. Dostupné z: <https://www.simplilearn.com/tutorials/c-tutorial/use-of-c-language>. [cit. 2024-07-08].
- RINI, Nicholas. *Advantages and Disadvantages of OOP in Java*. Online. Developer.com. 2023. Dostupné z: <https://www.developer.com/java/oop-advantages-disadvantages/>. [cit. 2023-11-12].
- ROLLER, Joshua. *Coding From 1849 to 2022: a Guide to The Timeline of Programming Languages*. Online. IEEE Computer Society. 2022. Dostupné z: <https://www.computer.org/publications/tech-news/insider-membership-news/timeline-of-programming-languages/>. [cit. 2023-11-25].
- ROUBALOVÁ, Eliška. *Java bez předchozích znalostí*. Brno: Computer Press, 2015. ISBN 978-80-251-4572-2.

ROUSE, Margaret. *Syntax*. Online. Techopedia. 2018, 6.12.2018. Dostupné z: <https://www.techopedia.com/definition/3959/syntax>. [cit. 2023-11-05].

ROUSE, Margaret. *C plus plus Programming Language*. Online. Techopedia. 2021. Dostupné z: <https://www.techopedia.com/definition/26184/c-plus-plus-programming-language>. [cit. 2023-11-15].

ROUSE, Margaret. *Procedural Language*. Online. Techopedia. 2017, 23.2.2017. Dostupné z: <https://www.techopedia.com/definition/8982/procedural-language>. [cit. 2023-11-05].

ROUSE, Margaret. *Variable*. Online. Techopedia. 2022. Dostupné z: <https://www.techopedia.com/definition/25647/variable-programming>. [cit. 2023-11-27].

ROUSE, Margaret. *Sementics*. Online. Techopedia. 2018, 11.1.2018. Dostupné z: <https://www.techopedia.com/definition/687/semantics-computing>. [cit. 2023-11-05].

RYAN, Mark. *Ada Lovelace, her Objection, Turing Tests and Universal Computing*. Online. Medium. 2019. Dostupné z: <https://medium.com/swlh/ada-lovelace-her-objection-e189717bd262>. [cit. 2024-07-07].

SAGAR, Paresh. *Swift vs Objective C: Which is Better for iOS Development?* Online. Excellent Webworld. 2023. Dostupné z: <https://www.excellentwebworld.com/swift-vs-objective-c/>. [cit. 2023-11-28].

SATYABRATA, Jena. *Difference between Byte Code and Machine Code*. Online. GeeksforGeeks. Uttar Prades: GeeksforGeeks, 2020. Dostupné z: <https://www.geeksforgeeks.org/difference-between-byte-code-and-machine-code/>. [cit. 2023-04-09].

SHASHOUA, Michael. *What Is PHP Used For?* Online. Codecademy Blog. 2021. Dostupné z: <https://www.codecademy.com/resources/blog/what-is-php-used-for/>. [cit. 2023-11-16].

SIMPLILEARN. *What is C Programming?* Online. Simplillearn. 2023, 13.2.2023. Dostupné z: <https://www.simplilearn.com/c-programming-article>. [cit. 2023-11-05].

SSHUBHAMSINGH10. *Difference between Compiled and Interpreted Language*. Online. GeeksforGeeks. Dostupné z: <https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/>. [cit. 2023-11-12].

STACKPATH. *What is a Web Application?* Online. StackPath. 2023. Dostupné z: <https://www.stackpath.com/edge-academy/what-is-a-web-application>. [cit. 2024-07-08].

TEKTUTORIALSHUB. *Type Systems in Programming Languages*. Online. TekTutorialsHub. 2022. Dostupné z: <https://www.tektutorialshub.com/programming/type-systems-in-programming-languages/>. [cit. 2024-07-05].

THOMPSON, Barbara. *What is .NET Framework? Explain Architecture*. Online. Guru99. 2023. Dostupné z: <https://www.guru99.com/net-framework.html>. [cit. 2023-11-15].

TUAMA, Daragh Ó. *What Is A Programming Language?* Online. Code institute. Dublin: Code institute, 2023. Dostupné z: <https://codeinstitute.net/global/blog/what-is-a-programming-language/>. [cit. 2023-06-17].

VAYOLA, Pradeep. *How Python Programs Works?* Online. DEV Community. 2016. Dostupné z: <https://dev.to/vayolapradeep/how-python-programs-works-15f>. [cit. 2023-11-15].

VOLLE, Adam. C++. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/C-computer-language>. [cit. 2023-11-13].

VOLLE, Adam. C++. Online. Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/C-computer-language>. [cit. 2023-11-13].

VOLLE, Adam. C++. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/C-computer-language>. [cit. 2023-11-15].

VOLLE, Adam. C++. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/C-computer-language>. [cit. 2023-11-13].

VOLLE, Adam. C. Online. Encyclopedia Britannica. 2023. Dostupné z: <https://www.britannica.com/technology/C-computer-language>. [cit. 2023-11-15].

WAGNER, Bill. *A tour of the C*. Online. Microsoft Learn. 2023. Dostupné z: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/#net-architecture>. [cit. 2023-11-15].

WILKINS, Jessica. *What is Computer Programming?* Online. Freecodecamp. 2021, 8.10.2021. Dostupné z: <https://www.freecodecamp.org/news/what-is-programming/>. [cit. 2023-11-05].

YADAV, Anita. *Go Programming Language (Introduction)*. Online. GeeksforGeeks. 2023. Dostupné z: <https://www.geeksforgeeks.org/go-programming-language-introduction/>. [cit. 2023-11-25].

YADAV, Shivashish. *5 Ways to Select the Best Web Development Language for Your Project*. Online. Nenalezený vydavatel. 2023. Dostupné z: <https://www.turing.com/kb/best-web-development-language-for-project>. [cit. 2023-11-28].

YASEEN, Awais. *Best Programming Language For Desktop Application Development*. Online. Best Coding Language. 2023. Dostupné z: <https://bestcodinglanguage.com/for-desktop-applications/>. [cit. 2024-07-08].

ZEZULA, Ladislav. *Výuka assembleru*. Online. Zezula.net. Zezula.net, 2003. Dostupné z: http://www.zezula.net/cz/teach/assembler_01.html. [cit. 2023-04-17].

Programovací jazyk. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk. [cit. 2023-04-16].

Apache Groovy. Online. Apache Groovy. United Kingdom: the Apache Groovy project, 2023. Dostupné z: <https://groovy.apache.org/index.html>. [cit. 2023-04-22].

Introduction to Java TM Technology. Online. Oracle. Austin, Texas, USA: Oracle, 2023. Dostupné z: <https://www.oracle.com/java/technologies/introduction-to-java.html#318>. [cit. 2023-04-22].

About the Java Technology. Online. ORACLE Java Documentation. Austin, Texas, USA: Oracle, 2022. Dostupné z: <https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>. [cit. 2023-04-22].

Co je Java? Online. Microsoft.com. Redmond, Washington, USA: Microsoft, 2023. Dostupné z: <https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-programming-language/>. [cit. 2023-04-22].

Babbageův počítač stroj. Online. Norgren. Brno: IMI International, 2023. Dostupné z: <https://www.norgren.com/cz/cs/marvellous-machines/marvellous-machines-in-history/babbage-computer>. [cit. 2023-04-17].

Co je to assembler? Online. IT-slovník.cz. ITslovník, 2022. Dostupné z: <https://it-slovník.cz/pojem/assembler>. [cit. 2023-04-17].

Interpreted vs Compiled Programming Languages: What's the Difference? Online. FreeCodeCamp. FreeCodeCamp, 2020. Dostupné z: <https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/>. [cit. 2023-04-09].

What is the difference between a compiled and an interpreted program? Online. Indiana: University Information Technology Services, 2021. Dostupné z: <https://kb.iu.edu/d/agsz>. [cit. 2023-04-09].

Computer programming. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://en.wikipedia.org/wiki/Computer_programming. [cit. 2023-04-09].

Computer program. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://en.wikipedia.org/w/index.php?title=Computer_program&oldid=1148837753. [cit. 2023-04-09].

Programming language. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: https://en.wikipedia.org/w/index.php?title=Programming_language&oldid=1148550195. [cit. 2023-04-09].

Algorithm. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2023. Dostupné z: <https://en.wikipedia.org/wiki/Algorithm>. [cit. 2023-04-09].

Seznam obrázků

Obrázek 1 - Vývoj nejpopulárnějších programovacích jazyků na GitHubu	35
Obrázek 2 - Nejžádanější programovací jazyky podle počtu pracovních míst	36