VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

POROVNÁNÍ JAZYKA PYTHON S JINÝMI PROGRAMOVACÍMI JAZYKY Z HLEDISKA SYNTAXE

Bakalářská práce

Autor práce: Oleh Strokan

Vedoucí práce: Ing. Marek Musil

Jihlava 2024

Vysoká škola polytechnická Jihlava

Tolstého 16, 586 01 Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Oleh Strokan**
Studijní program:	Aplikovaná informatika
Obor:	Aplikovaná informatika
Garant studijního programu:	Ing. Lenka Kuklišová Pavelková, Ph.D.
Název práce:	**Porovnání jazyka Python s jinými programovacími jazyky z hlediska syntaxe**
Vedoucí práce:	Ing. Marek Musil
Cíl práce:	Cílem práce je detailně nastudovat jazyk Python a porovnat ho s dalšími existujícími programovacími jazyky. Porovnání se zaměří na programátorské možnosti jazyka především z hlediska jeho syntaxe (přehlednost zápisu, snadnost zápisu, využitelnost v implementaci, atp.). K porovnání budou adekvátně zvoleny další 1 - 2 programovací jazyky.

Abstrakt

Tato práce pečlivě zkoumá jazyk Python a srovnává ho s ostatními existujícími programovacími jazyky z hlediska syntaxe. Cílem je analyzovat a porovnat programátorské možnosti Pythonu, s důrazem na přehlednost a snadnost zápisu kódu, jeho využitelnost při implementaci a další aspekty. Porovnání bude prováděno s ohledem na výběr dalších dvou programovacích jazyků, což umožní objektivní zhodnocení výhod a nevýhod syntaxe Pythonu ve srovnání s těmito jazyky.

Klíčová slova

C++; Java; Python; Rust; syntaxe jazyka; Typescript

Abstract

This work delves into a comprehensive examination of the Python language and its comparison with other existing programming languages in terms of syntax. The objective is to analyze and compare the programming capabilities of Python, focusing particularly on code readability, ease of writing, its implementation utility, and other pertinent aspects. The comparison will emphasize the selection of two additional programming languages, facilitating an objective evaluation and comparison of the advantages and disadvantages of Python's syntax in relation to these languages.

Keywords

C++; Java; language syntaxe; Python; Rust; Typescript

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Podle § 47b zákona o vysokých školách souhlasím se zveřejněním své práce podle Směrnice pro vedení, vypracování a zveřejňování závěrečných prací na VŠPJ, a to bez ohledu na výsledek obhajoby.

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé Zvolte položku. práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mě požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 9. dubna 2024

…….

Podpis studenta/ky

Poděkování

Rád bych vyjádřil svou upřímnou vděčnost všem, kteří přispěli k dosažení tohoto cíle a dokončení mé diplomové práce. Chtěl bych poděkovat mému vedoucímu práce Ing. Marku Musilovi a to ca cenné rady, podnětné diskuze a trpělivost během celého procesu psaní této práce. Vaše podpora a přínos byly neocenitelné a měly zásadní vliv na úspěšné dokončení této práce. Děkuji vám za vaši laskavost, pomoc a podporu.

Obsah

Seznam obrázků

Seznam zkratek

API	Application Programming Interface
TS	Typescript

Úvod

V dnešní digitální éře hrají programovací jazyky klíčovou roli ve vývoji softwaru a automatizaci procesů. Mezi mnoha existujícími jazyky se v poslední době značně prosadil Python díky své jednoduché syntaxi, širokému využití a rostoucí popularitě ve světě vývoje softwaru.

V současném kontextu, kde digitalizace proniká do všech oblastí lidské činnosti, je důležité zkoumat, jak programovací jazyky napomáhají vytváření sofistikovaných softwarových produktů a efektivní automatizaci procesů. Python jakožto jeden z nejrychleji rostoucích programovacích jazyků nabízí svou jednoduchou syntaxi a bohaté knihovny pro široké spektrum aplikací od webových aplikací až po strojové učení.

V dnešní digitální éře, kde je digitalizace stále více přítomná ve všech oblastech lidské činnosti, je důležité porozumět roli programovacích jazyků ve vývoji softwaru a automatizaci procesů. Jsem přesvědčen/a, že téma zkoumání výhod a možností Pythonu v tomto kontextu je aktuální a relevantní z několika důvodů. Za prvé, Python je jedním z nejrychleji rostoucích programovacích jazyků díky své jednoduchosti a všestrannosti. Jeho široké použití od webových aplikací až po strojové učení poskytuje obrovský potenciál pro vývoj sofistikovaných softwarových produktů a efektivní automatizaci procesů.

Téma programovacího jazyka Python jsem si vybral, protože mě zajímá, jak Pythonová syntaxe a knihovny přispívají k přehlednosti a efektivitě psaní kódu a implementaci různých aplikací. Chci lépe porozumět, jak Pythonové nástroje usnadňují vytváření komplexních softwarových systémů a jaké jsou jeho výhody a nevýhody ve srovnání s jinými populárními programovacími jazyky. Kromě toho mě motivovala i rostoucí popularita Pythonu ve světě vývoje softwaru a jeho vliv na moderní technologické trendy. Zároveň věřím, že analýza a srovnání Pythonu s dalšími jazyky mi poskytne užitečné poznatky pro mé profesní a akademické aktivity v oblasti softwarového inženýrství.

Cílem této bakalářské práce je provést důkladné nastudování syntaxe jazyka Python a provést porovnání programovacího jazyka Python s dalšími používanými programovacími jazyky. Tato práce bude řešit porovnání možností programových realizací provedených v jazyce Python a srovnávat řešení s programovacími jazyky jako jsou Java, C++, Rust a TypeScript. Práce se zaměří na různé aspekty jazyka, včetně syntaxe, přehlednosti zápisu, snadnosti použití v implementaci a dalších klíčových faktorů, které ovlivňují efektivnost a flexibilitu programování v daném jazyce. Práce ukazuje vhodnost použití programovacího jazyka Python a možnosti jeho syntaxe z hlediska přehlednosti v zápisu zdrojového kódu.

První část práce se bude věnovat detailnímu studiu jazyka Python, představení historie jazyka, předních vlastností a využití jazyka v praxi. Tato část bude poskytovat základní povědomí

o Pythonu jakožto významném hráči v oblasti programování a jeho klíčových rysů, které ho činí atraktivním pro široké spektrum programátorů. Následně budou vybrány a studovány další programovací jazyky, které budou použity k porovnání s Pythonem. Odůvodnění volby jazyků je součástí práce. Tato komparativní analýza umožní lépe porozumět jednotlivým jazykům a jejich přednostem a nedostatkům ve srovnání s Pythonem. Další část práce se zaměří na praktickou analýzu a porovnání jazyků prostřednictvím implementace konkrétních úkolů a scénářů. Bude provedena analýza syntaxe, srovnání využitelnosti v různých oblastech vývoje softwaru

a zhodnocení silných a slabých stránek každého jazyka. V závěrečné části práce budou prezentovány výsledky analýzy a závěry z porovnání jednotlivých jazyků. Na základě těchto výsledků budou formulovány doporučení a stanovena perspektiva dalšího vývoje v oblasti programovacích jazyků a jejich využití v praxi. Tímto přístupem se práce snaží poskytnout ucelený pohled na problematiku výběru programovacích jazyků a přispět k rozhodovacímu procesu výběru vhodného jazyka pro konkrétní projekty a aplikace.

1 Teoretická část

V teoretické části mé diplomové práce se zaměřím na klíčové koncepty související s programovacím jazykem Python a jeho srovnáním s dalšími existujícími jazyky. Budou vybrány jazyky dnes používané při vývoji aplikací.

Nejprve budu podrobněji zkoumat historii a vývoj programovacích jazyků obecně, abychom lépe porozuměli kontextu a významu jazyka Python v současném světě softwarového inženýrství. Dalším důležitým tématem, které bude probíráno, bude význam a využití programovacích jazyků v praxi, včetně jejich role v různých oblastech vývoje softwaru a jejich přínosu pro efektivní tvorbu a údržbu softwarových systémů. Budu se také zabývat důležitými koncepty a technologiemi souvisejícími s vývojem softwaru, jako je objektově orientované programování, funkcionální programování, správa paměti a další, a analyzovat, jak Python a vybrané další jazyky těchto konceptů využívají. Nakonec budu zkoumat současné trendy a perspektivy vývoje programovacích jazyků a jejich význam v kontextu moderního softwarového inženýrství a technologických inovací. Tato část bude klíčová pro pochopení významu porovnání jazyků a možných dopadů na budoucí vývoj softwarových aplikací.

V rámci porovnání programovacího jazyka Python s dalšími vybranými jazyky se budu zaměřovat zejména na TypeScript, Rust, Java a C++. TypeScript je významný zejména pro svou rostoucí popularitu v oblasti webového vývoje a mám s ním rozsáhlé zkušenosti. Rust mě zaujal svým zaměřením na bezpečnost a výkon a je zajímavým protějškem Pythonu, který se orientuje spíše na jednoduchost a flexibilitu. Klasické jazyky jako Java a C++ jsou stále základními pilíři softwarového inženýrství a jejich porovnání s Pythonem přinese cenný pohled na jejich silné a slabé stránky v různých kontextech vývoje aplikací. Porovnání jazyků bude provedeno formo posouzení programových implementací praktických problémů.

1.1 Návrh řešení

V této části práce plánuji nejprve podrobně vystěhovat vlastnosti a charakteristiky programovacího jazyka Python. Zaměřím se na jeho syntaxi, klíčové vlastnosti a typické scénáře použití v softwarovém inženýrství. Poté budu provádět porovnání syntaxe a praktického využití jazyka Python s dalšími vybranými jazyky, jako je TypeScript, Rust, Java a C++. Analyzovat budu jejich syntaxi, způsob práce s daty, manipulace s textem, práce se soubory a síťové operace. Pro srovnání a evaluaci budu využívat objektivní kritéria, která budou zahrnovat přehlednost zápisu, flexibilitu, výkon a efektivitu kódu. Tato kritéria budu aplikovat na konkrétní úkoly a scénáře, abych mohl posoudit, jak se jednotlivé jazyky chovají v praxi a jaké jsou jejich silné a slabé stránky.

Během analýzy budu pracovat s reálnými daty a kódy, a také budu konzultovat s komunitou a expertními uživateli jednotlivých jazyků. Tato interakce mi pomůže získat ucelený obraz o vlastnostech a použití jednotlivých jazyků v reálných projektech. Celkově se zaměřím na co nejkomplexnější a objektivní porovnání jazyků, abych mohl poskytnout užitečné informace a doporučení pro vývojáře a profesionály v oblasti softwarového inženýrství.

1.2 Jazyk Python, představení jazyka

Python je interpretovaný vysokoúrovňový programovací jazyk vytvořený Guidem van Rossumem a poprvé uvedený na trh v roce 1991. Matthes Eric v svém kurzu Python Crash Course (2019) tvrdí Název jazyka pochází ne z hada, ale z populárního komediálního pořadu 70. let - Monty Python's Flying Circus. Byl vyvinut jako jednoduchý a pohodlný programovací jazyk, který disponuje čitelnou syntaxí a zaměřuje se na zlepšení produktivity vývojářů a čitelnosti kódu.

Obr. 1: Logo programovacího jazyka Python
Zdroj: The advantages of learning Python *(2024)*

Historie vzniku
Guido van Rossum začal vyvíjet Python na konci 80. let, aby vytvořil náhradu za jazyk ABC. Chtěl vytvořit programovací jazyk s jednoduchou a srozumitelnou syntaxí. Hlavními principy návrhu Pythonu jsou čitelnost, jednoduchost používání a podpora různých programovacích paradigmat.

Požití Pythonu
Python je široce využíván v různých oblastech:

- Webové vývoj: Frameworky Django a Flask jsou používány pro tvorbu webových aplikací.
- Vědecké výpočty a analýza dat: NumPy, Pandas, Matplotlib - klíčové nástroje pro vědecký výzkum, analýzu dat a vizualizaci.
- Strojové učení a umělá inteligence: TensorFlow, PyTorch, Scikit-learn - populární knihovny pro vývoj strojového učení.
- Automatizace a správa systémů: Python se využívá pro tvorbu skriptů, automatizaci úkolů a správu systémů.

Verze Pythonu
Podle oficiální dokumentace Pythonu existují dvě hlavní verze: Python 2.x a Python 3.x. Python 2.x je stále podporován, ale doporučuje se používat Python 3.x kvůli lepší podpoře a novým možnostem. Přechod z Pythonu 2 na Python 3 může vyžadovat změny v syntaxi kódu kvůli určitým nekompatibilitám.

Podpora a společnosti používající Python:
Python má aktivní vývojářskou komunitu a obsáhlou dokumentaci. Podporuje se na různých operačních systémech a využívají ho velké společnosti, jako jsou Google, Facebook, Instagram, Spotify, Dropbox, Netflix. Používají Python pro vývoj a implementaci svých produktů díky jeho flexibilitě, široké škále knihoven a jednoduchosti použití.

Výhody a nevýhody Pythonu
Výhody:

- Jednoduchá a čitelná syntaxe.
- Široké množství knihoven a frameworků pro různé oblasti.
- Podpora různých platforem a operačních systémů.
- Široké využití ve vědě, webovém vývoji, strojovém učení.

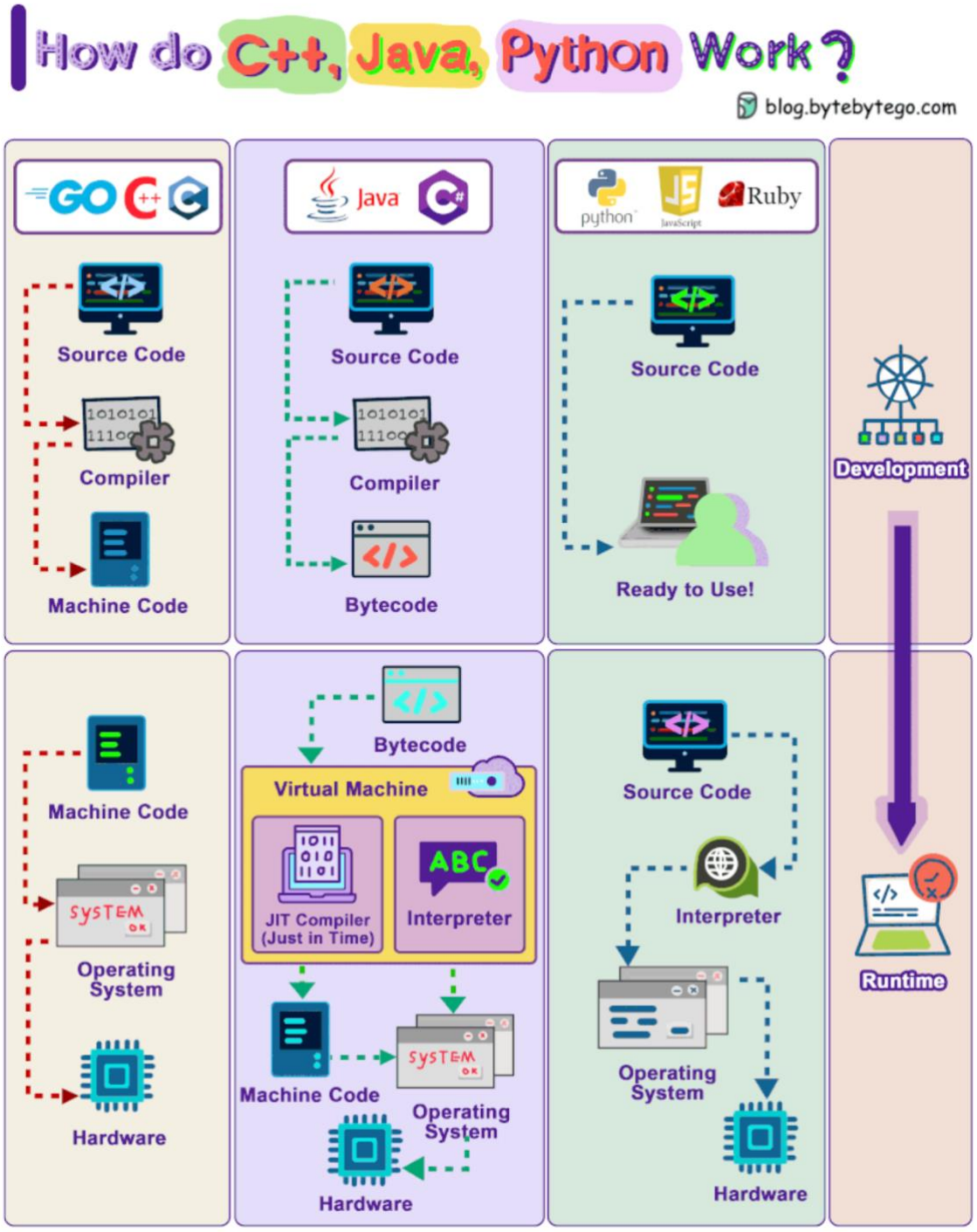
Nevýhody:

- Některé operace mohou být pomalejší než u některých kompilovaných jazyků.
- Ne všechny knihovny mohou být optimalizovány pro vysokou výkonost.
- Python nemusí být nejlepší volbou pro některé typy velmi náročných úkolů nebo pro práci s nízko-úrovňovými systémy.

Jak funguje Python do hloubky
Python funguje jako interpretovaný jazyk (viz obr. 2), což znamená, že kód Pythonu není přeložen do strojového kódu před spuštěním, ale interpretován řádek po řádku za běhu programu. Interpret přečte zdrojový kód Pythonu a převádí ho na instrukce strojového jazyka, které jsou vykonávány interpretací.

Dvěma nejpoužívanějšími implementacemi Pythonu jsou CPython a PyPy. CPython je standardní implementace Pythonu, která je napsaná v jazyce C a používá interpret na vykonávání kódu napsaného v syntaxi jazyka Python. Na druhou stranu PyPy je alternativní implementace Pythonu, která obsahuje JIT (Just-In-Time) kompilátor, který může v určitých případech zrychlit běh kódu. Oba tyto interprety čtou zdrojový kód Pythonu a převedou ho na instrukce strojového kódu, které jsou poté vykonány interpretem. Tento proces umožňuje spouštění programů jazyka Python na různých platformách, aniž by bylo nutné překompilovat kód pro každou z nich.

Python je silný a univerzální programovací jazyk, který je široce využíván v různých oblastech díky své jednoduchosti, flexibilitě a různorodosti knihoven.



Obr. 2: How do C++, Java, Python work?

Zdroj: blog.bytebytego.com (2023)

2 Výzkumná část

Proč je Python tak fascinující a proč se hodí pro výuku a porozumění základních principů programování? Syntaxe Pythonu je klíčovým prvkem, který z něj činí tak populární jazyk. V této části práce se zaměřím na přehledné a snadné příklady syntaxe Pythonu, které jsou vhodné pro výuku a demonstraci jeho programátorských možností.

2.1 Návrh řešení

Naše cesta začne základními principy proměnných a operátorů. Ukážeme si, jak deklarovat a manipulovat s proměnnými různých typů a jaké operátory jsou k dispozici pro různé operace v Pythonu. Přiblížíme si také podmínky a cykly, které jsou klíčové pro tvorbu programů. Další část bude věnována funkcím a způsobům definice a použití funkcí v Pythonu. Zde se zaměříme na to, jak efektivně využít funkce k usnadnění kódu a znovupoužitelnosti kódu. Dále se podíváme na datové struktury, jako jsou seznamy, slovníky a n-tice, a na jejich manipulaci. Ukážeme si, jak efektivně ukládat a pracovat s daty různých typů. Neopomeneme ani práci s výjimkami a soubory, což jsou klíčové koncepty pro robustní programování. Pokryjeme také nástroje jako je mapování a generátory, které Python nabízí pro zpracování dat a optimalizaci kódu.

Tyto příklady poslouží jako základ pro pochopení syntaxe Pythonu a jeho flexibilního použití v různých oblastech programování. Budeme se snažit ukázat, jakým způsobem lze tato pravidla syntaxe aplikovat na praktické úkoly a projekty.

Tato část práce nabídne ucelený pohled na syntaxi Pythonu prostřednictvím praktických ukázek a zároveň poskytne vhled do jeho univerzálnosti a srozumitelnosti pro začátečníky i pokročilé programátory.

2.2 Vyhledání literatury

Uvedená literatury byly vyhledána a použita v rámci realizace bakalářské práce. Budu používat tyto knižní zdroje a internetové zdroje.

Knižní zdroje

- Python

"Python Crash Course" od Eric Matthes (2019)

"Automate the Boring Stuff with Python" od Al Sweigart (2015)

- Java

"Head First Java" od Kathy Sierra a Bert Bates (2005)

"Effective Java" od Joshua Bloch (2018)

- C++

"Programming: Principles and Practice Using C++" od Bjarne Stroustrup (2014)

"The C++ Programming Language" od Bjarne Stroustrup (2013)

- Rust

"The Rust Programming Language" od Steve Klabnik a Carol Nichols (2018)

"Programming Rust" od Jim Blandy a Jason Orendorff (2019)

- TypeScript

"Programming TypeScript" od Boris Cherny (2020)

"Pro TypeScript: Application-Scale JavaScript Development" od Steve Fenton (2014)

Internetové (webové) zdroje

- Python

Oficiální dokumentace Pythonu: python.org

Real Python: realpython.com

- Java

Oficiální dokumentace Javy: docs.oracle.com

Baeldung: baeldung.com

- C++

cppreference.com: cppreference.com

LearnCpp.com: learncpp.com

- Rust

Rust by Example: rust-lang.org

The Rust Programming Language Blog: blog.rust-lang.org

- TypeScript

TypeScript Handbook: typescriptlang.org

TypeScript Deep Dive: basarat.gitbook.io/typescript

3 Realizace

3.1 Posouzení programovacích jazyků

V rámci této sekce se zaměříme na identifikaci rozdílů mezi programovacími jazyky a představení těchto rozdílů.

Na základě analýzy subjektivního názoru a osobních zkušeností lze konstatovat, že každý programovací jazyk má své výhody a nevýhody v závislosti na konkrétních potřebách a kontextu použití. Pokud je potřeba rychle vytvořit jednoduchý backend pro MVP (Minimum Viable Product), i) je *Python* často považován za vhodnou volbu díky své jednoduchosti a rychlosti vývoje. Python rovněž dobře slouží pro implementaci jednotlivých modulů systému, například pro služby pro odesílání e-mailů nebo upozornění. Při porovnání s jinými jazyky, jako je Java, se ukazuje, že každý jazyk má svá specifika a vhodnost v různých situacích. Java je široce využívána v korporátním prostředí díky své robustnosti a rozsáhlé komunitě. ii) Na druhou stranu, *TypeScript*, nadstavba jazyka *JavaScript*, nabízí statickou typovou kontrolu, což zvyšuje bezpečnost a robustnost kódu, zejména pro větší a rozsáhlejší projekty. TypeScript také zahrnuje moderní funkce jazyka, které usnadňují vývoj a udržování aplikací. iii) Stejně tak *Rust*, systémový jazyk, je známý pro svou vysokou výkonnost a bezpečnost. Rust se často využívá pro vývoj kritických aplikací, kde je klíčová spolehlivost a bezpečnost, jako jsou operační systémy nebo aplikace pro manipulaci s daty.

Zatímco Python může být vhodný pro rychlý vývoj, jazyk C++ jak říká Stroustrup, Bjarne v knize Programming: Principles and Practice Using C++ (2014) . se často využívá pro výkonné aplikace, které vyžadují optimalizaci paměti a výkonu. C++ je populární zejména v herním průmyslu a aplikacích vyžadujících nízko úrovňovou manipulaci s pamětí. Tento přístup nám umožní detailně analyzovat rozdíly mezi programovacími jazyky a poskytnout objektivní pohled na jejich vlastnosti a vhodnost v různých kontextech.

3.2 Můj názor

Na základě svého subjektivního názoru a osobní zkušenosti mohu říci, že Python je velmi dobrý, ale zároveň i velmi špatný programovací jazyk. Pokud potřebujeme rychle vytvořit jednoduchý backend pro MVP, pak je Python nejlepší volbou. Python také dobře vyhovuje pro jednotlivé moduly systému, jako je například služba pro odesílání e-mailů nebo upozornění. S jazykem Python mám již mnoholeté zkušenosti:

1. Před třemi lety jsem psal backendovou aplikaci v Pythonu 3, jejíž účel spočíval v tom, že získávala data z optimalizačního enginu, což byla mikroslužba vyvinutá v .NET a obsahovala mnoho algoritmů pro výpočty. Náš backend v Pythonu pouze posílal event message pomocí Kafka pro získání dat a pak je transformoval pro frontendovou aplikaci. Náš backend byl jednoduchou vrstvou mezi optimalizačním enginem a frontendovou aplikací.

2. Před dvěma lety jsem pracoval v Slovenské spořitelně, kde byl backend v Javě, a to je příklad, kde by Python nebyl vhodný. Měli jsme aplikaci o 380 000 řádcích kódu, která byla vyvíjena již několik let. Java byla lepší volbou, protože je stabilnější, méně dynamická a je snazší najít profesionální vývojáře v Javě, zatímco na Pythonu je hodně juniorů.

Pokud vybíráte programovací jazyk pro vaši aplikaci, je důležité zvážit její typ, termín dodání, počet lidí a finanční možnosti. Pro startupy je Python téměř vždy dobrá volba, protože startup potřebuje rychle ukázat hotový produkt, i když s chybami. Pro banky jsou nejlepší volby Java nebo mladší bratr C# .NET. Pro nové finanční aplikace bych použil TypeScript nebo případně Golang.

Ohledně Rustu a C++ mohu taky říci pár věcí. Tyto jazyky jsou si velmi podobné v tom, že umožňují kontrolu paměti a s dostatečnými znalostmi můžete dosáhnout vyššího výkonu. Mám zkušenosti s Rustem, protože jsem psal smart kontrakty pro blockchain[1]. A to je právě důvod, proč se tento jazyk učit. Pokud vás zajímá blockchain, pak polovina z toho, co je napsáno, je v jazyce Rust.

S C++ jsem se setkal především na univerzitě a při některých vlastních projektech jako je "build your own docker, build your own OS". Rust není na trhu populární jako java a python, a pro začátečníky může být obtížné najít práci, ale pro zkušené programátory je to ráj. Co se týče C++, řekl bych, že práce je, ale většinou to nejsou IT společnosti s vysokými platy. Ale pokud jde o vývoj her, pak je C++ vaším volbou. Také subjektivně, ale najít zkušeného vývojáře v C++ s znalostmi knihoven je mnohem snazší než v případě Rustu, prostě proto, že C++ je starší jazyk.

4.3 Porovnání mezi Python, Typescript a Rust

Naše rozhodnutí zahrnout TypeScript, Rust, Java a C++ do tohoto porovnání vychází z jejich významných pozic a jedinečných vlastností v oblasti programování.

- TypeScript, jako nadstavba JavaScriptu, nabízí statickou typovou kontrolu, což vede ke zvýšení bezpečnosti a robustnosti kódu. Jeho syntaxe a koncepty jsou podobné JavaScriptu, což usnadňuje přechod pro vývojáře. Výhody TypeScriptu jsou využívány v oblasti moderních webových aplikací a frameworků, kde se dynamická povaha JavaScriptu může stát zdrojem chyb a nejistot.

- Rust je systémový jazyk podobný jazykům C++/C, známý pro svou vysokou výkonnost a bezpečnost. Jeho silné zaměření na bezpečnost paměti a eliminaci nulové režie dělá z Rustu vhodnou volbu pro systémové programování, zejména pro kritické aplikace, jako jsou operační systémy, embedded zařízení a software pro manipulaci s daty.

- Java, jako jeden z nejpoužívanějších programovacích jazyků na světě, nabízí robustní a platformově nezávislou platformu pro vývoj softwaru. Je oblíbený zejména pro tvorbu velkých a rozsáhlých aplikací, včetně korporátních systémů a webových aplikací. Díky svému silnému ekosystému knihoven a frameworků má Java široké využití napříč různými odvětvími.

- C++, jako jeden z klasických a výkonných programovacích jazyků, je stále široce využíván ve vývoji softwaru, zejména v oblastech vyžadujících vysoký výkon a efektivitu, jako jsou herní

[1] **Blockchain** (z angl. "blokový řetězec") je v informatice speciální druh distribuované decentralizované databáze uchovávající neustále se rozšiřující počet záznamů, které jsou chráněny proti neoprávněnému zásahu jak z vnější strany, tak i ze strany samotných uzlů peer-to-peer sítě. (https://cs.wikipedia.org/wiki/Blockchain)

vývoj, systémové programování a nízkoúrovňový vývoj. Jeho robustní a flexibilní povaha ho činí důležitým subjektem pro porovnání s ostatními jazyky.
Rozhodli jsme se zahrnout TypeScript, Rust, Java a C++ do tohoto porovnání, abychom představili studentům různé paradigma programování a ukázali rozmanitost jazyků v různých kontextech a oblastech vývoje softwaru. To umožní studentům získat širší perspektivu a porozumění různým programovacím jazykům, jejich silným stránkám a použití v praxi. Tímto porovnáním se pokusíme poskytnout komplexní pohled na různé přístupy k programování a jejich využití v reálném světě.

3.3 Ukázkové příklady řešení

Nyní se zaměříme na ukázku syntaxe a rozdílů mezi jednotlivými programovacími jazyky na základě konkrétních příkladů. Následně se podíváme na jednoduché ukázky implementace základního aplikačního rozhraní (API). Tato cvičení nám umožní lépe porozumět specifikům každého jazyka a jeho použití v praxi. Pojďme se tedy ponořit do světa kódu a objevit, jak se jednotlivé jazyky liší a jak mohou být použity k vytváření služeb a aplikací.

Příklad č. 1: Základní “Hello world”

V prvním přikladu vypíšeme text v konzoli.

Python:

```
print("Hello, World!")
```

Python používá vestavěnou funkci *print()*, která vypisuje text v konzoli.

C++:

```
#include <iostream>

int main() {
    std::cout << "Hello, World!";
    return 0;
}
```

C++ definuje funkci *main()*, která je vstupním bodem programu. Používá objekt *std::cout*, který reprezentuje standardní výstup, a operátor << pro odeslání textu na standardní výstup.

Typescript:

```
console.log("Hello, World!");
```

TypeScript používá funkci console.log(), která je součástí JavaScriptu a TypeScriptu a vypisuje text v konzoli.

Rust:

```
fn main() {
    println!("Hello, World!");
}
```

Rust definuje funkci *main()*, která je vstupním bodem programu. Používá makro *println()*, které vypisuje text na konzoli.

Java:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Java definuje třídu HelloWorld s metodou *main()*, která je vstupním bodem programu. Používá metodu *System.out.println()*, která vypisuje text na konzoli.

Příklad č. 2: Deklarace proměnné a zobrazení hodnoty proměnné

Na dalším příkladu vytvoříme proměnnou a vypíšeme ji do konzole.

Python:

```
variable = "Hello, vspj!"
print(variable)
```

```
variable = 'Hello, vspj'
variable = 10 // only warning
```

Jak můžete vidět na tomto příkladu, v Pythonu je funkce *print* pro zobrazení dat v konzolovém okně aplikace. Také v Pythonu nejsou žádné příkazy pro deklaraci proměnných, jednoduše napíšete název proměnné rovná se nějaké hodnotě. Také v Pythonu neexistuje přesná typizace a můžete přepsat proměnnou na jinou hodnotu a nezískáte chybu, pouze varování. A to nejdůležitější. V Pythonu neexistují konstantní proměnné, můžete přepsat proměnnou kdekoli v kódu. To vše je opraveno balíčkem *pylint*, ale Python nemá tyto vlastnosti nativně.

C++:

```
#include <iostream>

int main() {
  string variable = "Hello, world!";
  scout << variable
  return 0;
}
```

V C++ máte klasickou syntaxi podobnou jazykům jako C. Je třeba zmínit, že ve většině těchto jazyků je potřeba funkce main, která spustí aplikaci.

Typescript:

```
const greeting: string = 'Hello, world!'
console.log(greeting)
```

```
let voteForTrump = 'Make America Great Again'
console.log(voteForTrump)
voteForTrump = 12309 // compile error
```

V TypeScriptu jsou k dispozici konstanty a přísná typizace. Navíc v TypeScriptu nemusíte psát typy dat, protože kompilátor může typ proměnné doplnit na základě hodnoty proměnné, jak je ukázáno ve druhém příkladu.

Rust:

```
fn main() {
  let variable = "Hello, world!";
  println!("{}", variable);
}
```

Rust má velmi lakonickou syntaxi podobnou Pythonu. Zde také, jako v TypeScriptu, může kompilátor doplnit návratový typ dat.

Java:

```
public class Main {
    public static void main(String[] args) {
    String variable = "Hello, world!";
    System.out.println(variable);
  }
}
```

V Javě stejně jako v jazycích podobných C je potřeba napsat funkci main.

Příklad č. 3: Datový typ Pole

V tomto příkladu vytvoříme pole, spočítáme jeho součet a vypíšeme ho do konzole.

Python:

```
array = [1, 2, 3, 4, 5]
sum_of_elements = sum(array)
print("Sum of elements in the array:", sum_of_elements)
```

V Pythonu můžete jednoduše vytvořit pole s různými datovými typy a použít funkci *sum* k vypočítání součtu prvků pole a vypsání toho do konzole.

C++:

```
#include <iostream>
#include <vector>
#include <numeric>

int main() {
  int array[] = {1, 2, 3, 4, 5};
  int sum_of_elements = accumulate(array.begin(), array.end(),
0);
  cout << "Sum of elements in the array: " << sum_of_elements <<
endl;
  return 0;
}
```

V C++ definujeme pole a pomocí funkce *accumulate* používáme begin a end pro určení začátku a konce

Typescript:

```
const array = [1, 2, 3, 4, 5]
const sumOfElements = array.reduce(
    (accumulator, currentValue) => accumulator + currentValue,
    0,
)
console.log('Sum of elements in the array:', sumOfElements)
```

V Typescript existuje podobná funkce nazvaná reduce. Tuto funkci můžeme použít k akumulaci hodnot každého prvku a výpisu pomocí běžné funkce console.log.

Rust:

```
fn main() {
  let array = [1, 2, 3, 4, 5];
  let sum_of_elements: i32 = array.iter().sum();
  println!("Sum of elements in the array: {}", sum_of_elements);
}
```

V Rustu můžete vytvořit pole a neuvádět typy, Rust si je sám odvodí a spočítá součet pomocí funkce sum.

Java (8 or later):

```
import java.util.Arrays;

  public class Main {
  public static void main(String[] args) {
  int[] array = {1, 2, 3, 4, 5};
  int sumOfElements = Arrays.stream(array).sum();
  System.out.println("Sum of elements in the array: " +
sumOfElements);
```

```
}
}
```

V Javě 8 můžeme použít funkci sum(), ale před touto verzí by bylo potřeba použít klasický cyklus.

Příklad č. 4: Použití funkce a konstrukce cyklu

Python (requests):

```python
import requests

def send_requests():
for i in range(1, 6):
response = requests.get(f"https://api.example.com/data/{i}")
print(f"Response for request {i}: {response.status_code}")

send_requests()
```

V následujícím příkladu používáme knihovnu *requests* k odesílání dotazů na externí API. Inicializujeme funkci, ve které je cyklus pomocí příkazu *for*. V cyklu vytváříme proměnnou response, které přiřadíme hodnotu vrácenou z externího API, a poté tuto hodnotu vypisujeme do konzole.

C++ (cURL):

```cpp
#include <iostream>
#include <curl/curl.h>

size_t writeCallback(void *ptr, size_t size, size_t nmemb,
std::string *data) {
  data->append((char*)ptr, size * nmemb);
  return size * nmemb;

void sendRequests() {
  CURL *curl;
  CURLcode res;
  std::string data;

  for (int i = 1; i <= 5; ++i) {
    curl = curl_easy_init();
    if (curl) {
      curl_easy_setopt(curl, CURLOPT_URL,
"https://api.example.com/data/" + std::to_string(i));
      curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
writeCallback);
      curl_easy_setopt(curl, CURLOPT_WRITEDATA, &data);
      res = curl_easy_perform(curl);
```

```
      std::cout << "Response for request " << i << ": " << res
<< std::endl;ij
      curl_easy_cleanup(curl);
    }
  }
}

int main() {
  sendRequests();
  return 0;
}
```

V C++ používáme balíček cURL k interakci s externími API. Funkce *curl_easy_setopt* inicializuje konfiguraci dotazu.

Typescript (fetch):

```
async function sendRequests(): Promise<void> {
  for (let i = 1; i <= 5; i++) {
    const response = await
fetch(`https://api.example.com/data/${i}`);
      console.log(`Response for request ${i}:
${response.status}`);
  }
}

sendRequests();
```

V Typescriptu používáme vestavěnou funkci fetch pro dotazy, a vypadá to podobně jako v Pythonu

Rust (reqwest):

```
use reqwest;

async fn send_requests() -> Result<(), reqwest::Error> {
  for i in 1..=5 {
  let response =
reqwest::get(format!("https://api.example.com/data/{}",
i)).await?;
  println!("Response for request {}: {}", i, response.status());
}
Ok(())
}

#[tokio::main]
async fn main() -> Result<(), reqwest::Error> {
  send_requests().await?;
  Ok(())
}
```

V Rustu inicializujeme funkci, ve které je cyklus, a v němž je zajímavý zápis čísel od 1 do 5 pomocí 1...5. Toto zajímavé řešení jsem nenašel v žádném z dalších uvedených jazyků. Také voláme funkci v main

Java (HttpClient):

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.util.concurrent.CompletableFuture;

public class Main {

  public static void sendRequests() {
    HttpClient client = HttpClient.newHttpClient();
    for (int i = 1; i <= 5; i++) {
      String url = "https://api.example.com/resource/" + i;
      HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(url))
        .build();

      CompletableFuture<Void> future = client.sendAsync(request,
HttpResponse.BodyHandlers.ofString())

        .thenApply(HttpResponse::statusCode)
        .thenAccept(statusCode -> System.out.println("Request "
+ i + " completed with status code: " + statusCode))
        .exceptionally(throwable -> {
          System.err.println("Request " + i + " failed: " +
throwable.getMessage());
          return null;
        });

      future.join();
    }
  }

  public static void main(String[] args) {
  sendRequests();
}
}
```

V Javě používáme balíček *HttpClient* a také vytváříme instanci třídy klienta, kterou budeme používat v cyklu

Příklad č. 5: Spojové seznamy

Python:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage:
if __name__ == "__main__":
    linked_list = LinkedList()
    linked_list.append(1)
    linked_list.append(2)
    linked_list.append(3)
    linked_list.display()
```

C++:

```cpp
#include <iostream>

struct Node {
    int data;
    Node* next;
```

```
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;
public:
    LinkedList() : head(nullptr) {}
    void append(int data) {
        Node* new_node = new Node(data);
        if (!head) {
            head = new_node;
            return;
        }
        Node* last_node = head;
        while (last_node->next) {
            last_node = last_node->next;
        }
        last_node->next = new_node;
    }
    void display() {
        Node* current = head;
        while (current) {
            std::cout << current->data << " -> ";
            current = current->next;
        }
        std::cout << "None" << std::endl;
    }
};

int main() {
    LinkedList linked_list;
    linked_list.append(1);
    linked_list.append(2);
    linked_list.append(3);
    linked_list.display();
    return 0;
}
```

Typescript:

```
class Node {
    data: number;
    next: Node | null;
    constructor(data: number) {
        this.data = data;
        this.next = null;
    }
```

```
}

class LinkedList {
    head: Node | null;
    constructor() {
        this.head = null;
    }

    append(data: number): void {
        const new_node = new Node(data);
        if (!this.head) {
            this.head = new_node;
            return;
        }
        let last_node = this.head;
        while (last_node.next) {
            last_node = last_node.next;
        }
        last_node.next = new_node;
    }

    display(): void {
        let current = this.head;
        while (current) {
            process.stdout.write(`${current.data} -> `);
            current = current.next;
        }
        console.log("None");
    }
}

const linkedList = new LinkedList();
linkedList.append(1);
linkedList.append(2);
linkedList.append(3);
linkedList.display();
```

Rust:

```
struct Node {
    data: i32,
    next: Option<Box<Node>>,
}

impl Node {
    fn new(data: i32) -> Node {
        Node { data, next: None }
    }
```

```
}

struct LinkedList {
    head: Option<Box<Node>>,
}

impl LinkedList {
    fn new() -> LinkedList {
        LinkedList { head: None }
    }

    fn append(&mut self, data: i32) {
        let new_node = Box::new(Node::new(data));
        let mut current_node = &mut self.head;
        while let Some(ref mut node) = current_node {
            current_node = &mut node.next;
        }
        *current_node = Some(new_node);
    }

    fn display(&self) {
        let mut current_node = &self.head;
        while let Some(node) = current_node {
            print!("{} -> ", node.data);
            current_node = &node.next;
        }
        println!("None");
    }
}

fn main() {
    let mut linked_list = LinkedList::new();
    linked_list.append(1);
    linked_list.append(2);
    linked_list.append(3);
    linked_list.display();
}
```

Java:

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
```

```
}

class LinkedList {
    Node head;

    LinkedList() {
        this.head = null;
    }

    void append(int data) {
        Node new_node = new Node(data);
        if (this.head == null) {
            this.head = new_node;
            return;
        }
        Node last_node = this.head;
        while (last_node.next != null) {
            last_node = last_node.next;
        }
        last_node.next = new_node;
    }

    void display() {
        Node current = this.head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("None");
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.append(1);
        linkedList.append(2);
        linkedList.append(3);
        linkedList.display();
    }
}
```

Na příkladu LinkedListu můžeme vidět, jak se tyto jazyky podobají svým konstrukcím a názvům základních metod

Příklad č. 6: (Více)vláknové zpracování

Vícevláknové zpracování: Mezi jazyky, které jste zmínil, podporují všechny vícevláknové zpracování v různé míře:

Python: Podporuje vícevláknové zpracování pomocí modulu threading. Kvůli globálnímu interpretátorovému zámků (GIL) je však vícevláknové zpracování v Pythonu vhodnější pro úlohy založené na vstupu/výstupu než pro úlohy založené na procesoru.

C++: Plně podporuje vícevláknové zpracování pomocí knihovny <thread> v normě C++11, která poskytuje třídy a funkce pro vytváření a správu vláken.

TypeScript: Protože se TypeScript překládá do JavaScriptu, běží na běhovém prostředí JavaScriptu které je výchozím nastavením jednovláknové. Nicméně TypeScript může stále využívat vícevláknové zpracování prostřednictvím webových pracovníků v prostředí prohlížeče nebo pomocí vláken pracovníků Node.js v serverových aplikacích.

Rust: Poskytuje robustní podporu pro paralelní a vícevláknové zpracování pomocí svého systému vlastnictví a modulu std::thread.

Java: Silně podporuje vícevláknové zpracování pomocí vestavěné třídy *java.lang.Thread* a nástrojů v balíčku java.util.concurrent. Konkurenční nástroje v Javě nabízejí sadu nástrojů pro správu vláken, synchronizačních primitiv a abstrakcí na vyšší úrovni pro souběžné programování.

Na základě těchto informací mohou být všechny uvedené jazyky použity pro vícevláknové zpracování, ale přístupy a knihovny pro zacházení s konkurencí se mohou lišit.

Python:

```
import threading
import time

def thread_function(name):
print("Thread", name, "started")
time.sleep(2)
print("Thread", name, "finished")

threads = []
for i in range(5):
thread = threading.Thread(target=thread_function, args=(i,))
threads.append(thread)
thread.start()

for thread in threads:
thread.join()

print("All threads have finished")
```

C++:

```
include <iostream>
#include <thread>
#include <vector>
```

```
void threadFunction(int id) {
  std::cout << "Thread " << id << " started" << std::endl;
  std::this_thread::sleep_for(std::chrono::seconds(2));
  std::cout << "Thread " << id << " finished" << std::endl;
}

int main() {
  std::vector<std::thread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.emplace_back(threadFunction, i);
  }

  for (auto& thread : threads) {
    thread.join();
  }

  std::cout << "All threads have finished" << std::endl;  return
0; }
```

Typescript:

```
const threadFunction = (id: number) => {
    console.log(`Thread ${id} started`)
    setTimeout(() => {
        console.log(`Thread ${id} finished`)
    }, 2000)
}

const threads = []
for (let i = 0; i < 5; i++) {
    threads.push(
        new Promise((resolve) => {
            threadFunction(i)
            resolve(() => console.log(`Thread ${i} has
finished`))
        }),
    )
}

Promise.all(threads).then(() => {
    console.log('All threads have finished')
})
```

Rust:

```
use std::thread;
use std::time::Duration;
```

```rust
fn thread_function(id: usize) {
  println!("Thread {} started", id);
  thread::sleep(Duration::from_secs(2));
  println!("Thread {} finished", id);
}

fn main() {
  let mut handles = vec![];
  for i in 0..5 {
    let handle = thread::spawn(move || {
      thread_function(i);
    });
    handles.push(handle);
  }

  for handle in handles {
    handle.join().unwrap();
  }
  println!("All threads have finished");
}
```

Java:

```java
class MyThread extends Thread {
  private int id;

  public MyThread(int id) {
  this.id = id;
}

public void run() {
  System.out.println("Thread " + id + " started");
  try {
    Thread.sleep(2000);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  System.out.println("Thread " + id + " finished");
}
}

public class Main {
  public static void main(String[] args) {
  Thread[] threads = new Thread[5];
  for (int i = 0; i < 5; ++i) {
  threads[i] = new MyThread(i);
  threads[i].start();
}
```

```
try {
  for (Thread thread : threads) {
    thread.join();
  }
} catch (InterruptedException e) {
  e.printStackTrace();
}
System.out.println("All threads have finished");
}}
```

3.4 Reálný příklad ze softwarového vývoje

Nyní vám představím konkrétní příklad z reálného světa, kterým je funkce pro měkké uzamčení dnů výletů v aplikaci.

Tato ukázka slouží k měkkému uzamčení dnů v aplikaci. Měkké uzamčení znamená dočasné označení dnů jako uzamčených, ale ne zcela trvale. V průběhu zpracování může být měkké uzamčení zrušeno a dny mohou být opět volně přístupné k úpravám a manipulaci. Jsou zde dvě funkce: První funkce je zodpovědná za přijetí data a v cyklu zaznamenává pomocí logování, vytváří asynchronní proměnné, které jsou výsledkem volání druhé funkce. Druhá funkce získává data z databáze po částech kvůli jejich velkému počtu, a proto je dobře použít generátory. Poté voláme repozitář, který mění tourday s lockState: LockState.SoftLock. Poté posíláme data do naší metrické služby, aby si zákazník mohl uvědomit, kolik má zablokovaných tourDay, a nakonec vracíme id všech zablokovaných tourday.

Python:

```
async def soft_lock_tour_days(date=start_of_today().isoformat())
-> List[TourDay['id']]:
locked_tour_day_ids: List[TourDay['id']] = []

async for unlocked_tour_days in
self.get_top_chunk_of_unlocked_tour_days_before(date):
tour_day_ids_to_lock = [to_id(tour_day) for tour_day in
unlocked_tour_days]

await self.repository.update_many_by_ids(tour_day_ids_to_lock, {
  'lock_state': LockState.SoftLock,
})

self.metric_service.tour_days.soft_locked(len(tour_day_ids_to_lo
ck))
locked_tour_day_ids.extend(tour_day_ids_to_lock)

return locked_tour_day_ids

async def get_top_chunk_of_unlocked_tour_days_before(before:
ISO8601NoTime) -> AsyncGenerator:
```

```
  while True:
tour_days = await
self.repository.find_many_unlocked_before_date(before, {
  'offset': 0,
  'limit': TOUR_DAY_CHUNK_SIZE,
})

if not tour_days:
  break

yield tour_days
```

C++:

```
// Soft lock tour days
std::vector<TourDayId> softLockTourDays(const std::string& date
= startOfToday()) {
  std::vector<TourDayId> lockedTourDayIds;

  for (const auto& unlockedTourDays :
getTopChunkOfUnlockedTourDaysBefore(date)) {
    std::vector<TourDayId> tourDayIdsToLock;
    for (const auto& tourDay : unlockedTourDays) {
      tourDayIdsToLock.push_back(toId(tourDay));
    }

    repository.updateManyByIds(tourDayIdsToLock, {
      "lock_state": LockState::SoftLock,
    });

    metricService.tourDays.softLocked(tourDayIdsToLock.size());
    lockedTourDayIds.insert(lockedTourDayIds.end(),
tourDayIdsToLock.begin(), tourDayIdsToLock.end());
  }

  return lockedTourDayIds;
}

// Get top chunk of unlocked tour days before
std::vector<TourDay> getTopChunkOfUnlockedTourDaysBefore(const
std::string& before) {
  std::vector<TourDay> tourDays;
  while (true) {
    auto unlockedTourDays =
repository.findManyUnlockedBeforeDate(before, {
      "offset": 0,
      "limit": TOUR_DAY_CHUNK_SIZE,
    });
```

```
    if (unlockedTourDays.empty()) {
      break;
    }

    tourDays.insert(tourDays.end(), unlockedTourDays.begin(),
unlockedTourDays.end());
  }

  return tourDays;
}
```

Typescript:

```
softLockTourDays = async (
    date = toISO8601NoTimeUTC(startOfToday()),
): Promise<Array<TourDay['id']>> => {
    this.logger.info('Soft locking tour days')

    const lockedTourDayIds: Array<TourDay['id']> = []

    for await (const unlockedTourDays of
this.getTopChunkOfUnlockedTourDaysBefore(date)) {
        const tourDayIdsToLock = unlockedTourDays.map(toId)

        await this.repository.updateManyByIds(tourDayIdsToLock,
{
            lockState: LockState.SoftLock,
        })


this.metricService.tourDays.softLocked(tourDayIdsToLock.length)
        lockedTourDayIds.push(...tourDayIdsToLock)
    }

    return lockedTourDayIds
}

private async *getTopChunkOfUnlockedTourDaysBefore(before:
ISO8601NoTime) {
    while (true) {
        const { items: tourDays } = await
this.repository.findManyUnlockedBeforeDate(before, {
            // The offset is always 0 since we are mutating tour
days from the top
            offset: 0,
            limit: TOUR_DAY_CHUNK_SIZE,
        })

        if (tourDays.length === 0) break

        yield tourDays
    }
}
```

Rust:

```
async fn soft_lock_tour_days(date: String) -> Vec<u32> {
  println!("Soft locking tour days");
```

```
  let mut locked_tour_day_ids = Vec::new();

  let mut tour_days_stream =
get_top_chunk_of_unlocked_tour_days_before(date).await;

  while let Some(unlocked_tour_days) =
tour_days_stream.next().await {
  let tour_day_ids_to_lock: Vec<u32> =
unlocked_tour_days.iter().map(|tour_day| tour_day.id).collect();

  update_many_by_ids(&tour_day_ids_to_lock,
LockState::SoftLock).await;


metric_service.tour_days.soft_locked(tour_day_ids_to_lock.len())
;

  locked_tour_day_ids.extend(tour_day_ids_to_lock);
}

locked_tour_day_ids
}

async fn get_top_chunk_of_unlocked_tour_days_before(before:
String) -> impl Stream<Item = Vec<TourDay>> {
  let mut offset = 0;

  futures::stream::try_unfold(offset, move |offset| async move {
    let result =
repository.find_many_unlocked_before_date(before.clone(),
offset, TOUR_DAY_CHUNK_SIZE).await?;

    if result.items.is_empty() {
  return Ok(None);
}

let tour_days = result.items;

Some((tour_days, offset + TOUR_DAY_CHUNK_SIZE))
})
}
```

Java:

```
public List<Integer> softLockTourDays(String date) {
  System.out.println("Soft locking tour days");
```

```
  List<Integer> lockedTourDayIds = new ArrayList<>();

  Stream<List<TourDay>> tourDaysStream =
getTopChunkOfUnlockedTourDaysBefore(date);

  tourDaysStream.forEach(unlockedTourDays -> {
    List<Integer> tourDayIdsToLock = new ArrayList<>();
    unlockedTourDays.forEach(tourDay ->
tourDayIdsToLock.add(tourDay.getId()));

    updateManyByIds(tourDayIdsToLock, LockState.SoftLock);

    metricService.tourDays.softLocked(tourDayIdsToLock.size());

    lockedTourDayIds.addAll(tourDayIdsToLock);
  });

  return lockedTourDayIds;
}

private Stream<List<TourDay>>
getTopChunkOfUnlockedTourDaysBefore(String before) {
  int offset = 0;

  return Stream.generate(() -> {
    List<TourDay> tourDays =
repository.findManyUnlockedBeforeDate(before, offset,
TOUR_DAY_CHUNK_SIZE).getItems();

    if (tourDays.isEmpty()) {
      return null;
    }

    offset += TOUR_DAY_CHUNK_SIZE;

    return tourDays;
  }).takeWhile(tourDays -> tourDays != null);
}
```

4 Výsledky a diskuze

Vyhodnocení vhodnosti nasazení programovacích jazyků: Python, TypeScript a Rust jsou jazyky, které mají své vlastní silné stránky a oblasti použití. Každý z nich se vyznačuje určitými vlastnostmi a je vhodný pro různé účely.

Python je obvykle volbou pro

- Rychlý vývoj: Jeho jednoduchý a čitelný kód jej činí ideálním pro rychlý vývoj a prototypování.
- Data Science a analýzu dat: Je oblíbený pro práci s daty díky knihovnám jako NumPy, Pandas a Matplotlib.
- Webový vývoj: Frameworky jako Django a Flask jsou často používány pro vytváření webových aplikací.

TypeScript vyniká v

- Velkých klientských aplikacích: Je silným nástrojem pro psaní rozsáhlých a robustních aplikací na straně klienta s výhodou statické typové kontroly.
- Použitelnost s JavaScriptem: TypeScript je rozšířením JavaScriptu, což umožňuje postupný přechod na statické typování.
- Přehlednost kódu: Typová kontrola a vysvětlující dokumentace činí kód srozumitelnějším.
- Rust se uplatňuje v těchto problematických oblastech:
- Systémové programování: Pro nízkoúrovňové úkoly nebo programování na úrovni systému, jako je operační systém nebo hardware, díky svému silnému zabezpečení a výkonu.
- Bezpečnost a paralelnost: Rust nabízí kontrolu paměti během kompilace, což pomáhá předcházet chybám spojeným s pamětí. Tato vlastnost je užitečná pro paralelní programování.
- Výkon a spolehlivost: Je vhodný pro aplikace, kde je klíčový výkon a spolehlivost.

Výběr mezi těmito jazyky závisí na specifických potřebách projektu. Python je často volbou pro rychlý vývoj a práci s daty. TypeScript je skvělý pro rozsáhlé klientské aplikace a Rust pro systémové programování, kde je klíčová výkonnost a bezpečnost.

5 Závěr

Závěr této práce představuje komplexní zhodnocení porovnání jazyka Python s dalšími významnými programovacími jazyky jako TypeScript a Rust. Tato studie se zaměřila na detailní analýzu syntaxe a programátorských možností těchto jazyků s důrazem na přehlednost, snadnost použití a implementaci.

Python, jako vysoceúrovňový, interpretovaný jazyk, vyniká svou jednoduchostí a čitelností syntaxe. Je široce využíván v oblastech webového vývoje, vědeckých výpočtů, umělé inteligence a automatizace díky své flexibilitě a rozsáhlým knihovnám. TypeScript, jako nadstavba JavaScriptu, přináší statickou typovou kontrolu a zvyšuje bezpečnost kódu. Jeho použití je obzvláště významné v oblasti moderního webového vývoje, kde dynamická povaha JavaScriptu může být nevýhodou. Rust, systémový jazyk, vyniká v oblastech vysoce výkonných aplikací a systémového programování díky své bezpečnosti paměti a efektivitě. Je vhodný pro aplikace, které vyžadují kritickou bezpečnost a rychlost.

Porovnání těchto jazyků ukázalo jejich jedinečné vlastnosti a aplikace. Zároveň podtrhlo různé přístupy a filozofie, které stojí za jejich návrhem a vývojem. Každý jazyk má své výhody a nevýhody a vhodnost použití záleží na specifických potřebách a cílech projektu

Tato práce poskytuje ucelený pohled na syntaktické a programátorské vlastnosti těchto jazyků. Pomáhá porozumět, jak různé jazyky nabízejí různé způsoby přístupu k programování a jakým způsobem mohou být efektivně využity v různých oblastech vývoje softwaru.

Hodnotím svoji práci pozitivně. Realizace tématu mě velmi zaujala a líbila se mi. Získal/a jsem hlubší pochopení Pythonu a jeho využití ve vývoji softwaru a automatizaci procesů. Práce na tomto projektu mi umožnila prozkoumat různé aspekty Pythonu a porovnat je s jinými programovacími jazyky, což bylo velmi poučné.

Jsem spokojen/a s dosaženými výsledky, ale věřím, že existují oblasti, které lze dále zkoumat a rozvíjet. Několik námětů na další rozšíření zahrnuje:

1. Pokročilé porovnání s dalšími jazyky: Rozšíření analýzy o další programovací jazyky, jako je Rust, TypeScript, C++, Java, a detailnější srovnání jejich výhod a nevýhod v konkrétních aplikacích.
2. Studium konkrétních oblastí aplikací Pythonu: Detailní zkoumání konkrétních oblastí využití Pythonu, jako je strojové učení, data science, web development atd.
3. Zkoumání nových verzí a aktualizací: Sledování vývoje Pythonu a jeho knihoven, a zkoumání nových funkcí a aktualizací.
4. Práce s komunitou: Zapojení do komunity Pythonu, účast na konferencích, workshopech a diskuzích pro sdílení znalostí a zkušeností s dalšími vývojáři.

Tyto nápady by mohly přinést další hloubku a šíři do práce a rozvinout ji do ještě komplexnějšího a relevantnějšího projektu v oblasti vývoje softwaru a automatizace procesů.

V závěru lze říct, že v dnešním dynamickém světě softwaru je pochopení různorodosti programovacích jazyků klíčem k úspěchu. Tato práce přispívá k širšímu pochopení a hodnocení jazyků Python, TypeScript a Rust, a vyzdvihuje jejich jedinečné vlastnosti a vhodnost použití v různých kontextech vývoje softwaru.

Seznam použité literatury

BLANDY, Jim, a Jason Orendorff. Programming Rust. O'Reilly Media, 2019. [cit. 2022-19-03]. Dostupné z: http://bzz.wallizard.com:8081/share/books/RUST/Programming%20Rust%202nd%20Edition.pdf

BLOCH, Joshua. Effective Java. Addison-Wesley, 2018. [cit. 2022-21-03]. Dostupné z: https://kea.nu/files/textbooks/new/Effective%20Java%20%282017%2C%20Addison-Wesley%29.pdf

ByteByteGo Newsletter [online]. [cit. 2024-04-07]. Dostupné z: blog.bytebytrego.com

FENTON, Steve. Pro TypeScript: Application-Scale JavaScript Development. Apress, 2014. ISBN 978-1430267911.

CHERNY, Boris. Programming TypeScript. O'Reilly Media, 2020. [cit. 2023-10-03]. Dostupné z: https://download.e-bookshelf.de/download/0013/3982/26/L-G-0013398226-0042821638.pdf

KLABNIK, Steve, a Carol Nichols. The Rust Programming Language. No Starch Press, 2018. [cit. 2024-10-03]. Dostupné z https://edu.anarcho-copy.org/Programming%20Languages/Rust/rust-programming-language-steve-klabnik.pdf

MATTHES, Eric. Python Crash Course. No Starch Press, 2019. [cit. 2024-10-03]. Dostupné z: https://ehmatthes.github.io/pcc/

SIERRA, Kathy, a Bert Bates. Head First Java. O'Reilly Media, 2005. [cit. 2024-07-03]. Dostupné z: https://www.rcsdk12.org/cms/lib/NY01001156/Centricity/Domain/4951/Head_First_Java_Second_Edition.pdf

STROUSTUP, Bjarne. Programming: Principles and Practice Using C++. Addison-Wesley, 2014. [cit.2024-23-02]. Dostupné z: https://dl.icdst.org/pdfs/files3/fef0590f02fa06bb42cba558fbc9e51c.pdf

STROUSTUP, Bjarne. The C++ Programming Language. Addison-Wesley, 2013. [cit. 2024-19-02]. Dostupné z: https://chenweixiang.github.io/docs/The_C++_Programming_Language_4th_Edition_Bjarne_Stroustrup.pdf

SWEIGART, Al. Automate the Boring Stuff with Python. No Starch Press, 2015. [cit. 2024-13-03]. Dostupné z: https://automatetheboringstuff.com

How C++, Java, and Python Work Under the Hood? [online]. LAM, Sahn. 2023 [cit. 2024-04-07]. Dostupné z: https://www.linkedin.com/posts/sahnlam_how-c-java-and-python-work-under-the-activity-7123514495817154563-6BkN?utm_source=share&utm_medium=member_desktop

The advantages of learning Python. Online. GUSTAVO, João. Analytics Vidhya, 2021. Dostupné z: https://medium.com/analytics-vidhya/the-advantages-of-learning-python-4401a185c053. [cit. 2024-04-07].

Přílohy

Projekt se zdrojovými kódy – *zdrojove_kody.zip*.