Automatizace hodnocení Python kódu ve výuce

Jan Vaněk

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan Vaněk**
Osobní číslo:	**A21016**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Automatizace hodnocení Python kódu ve výuce**
Téma práce anglicky:	**Automated Evaluation of Python Code in Computer Science Education**

Zásady pro vypracování

1. Prostudujte dostupné metody a nástroje vhodné pro automatizované hodnocení Python kódu v kontextu vzdělávání.
2. Analyzujte výhody a nevýhody automatizovaného hodnocení oproti manuálnímu přístupu.
3. Navrhněte vlastní flexibilní a rozšiřitelné řešení pro automatizované hodnocení Python kódu.
4. Vytvořte několik typových cvičení ilustrujících automatizované hodnocení.
5. Zhodnoťte kvalitu navrženého řešení a poskytněte doporučení pro jeho použití v praxi.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SUMMERFIELD, Mark, 2010. *Programming in Python 3: A Complete Introduction to the Python Language*. B.m.: Addison-Wesley Professional.
2. JAWORSKI, Michal and Tarek ZIADÉ, 2021. *Expert Python Programming – Fourth Edition: Master Python by Learning the Best Coding Practices and Advanced Programming Concepts*
3. OKKEN, Brian, 2022. *Python Testing with pytest*. B.m.: Pragmatic Bookshelf.
4. RAGHAVENDRA, Sujay, 2021. *Python Testing with Selenium: Learn to Implement Different Testing Techniques Using the Selenium Webdriver*.
5. GUNDECHA, Unmesh, 2015. *Learning Selenium Testing Tools with Python*. B.m.: CreateSpace.
6. CHANDRASEKARA, Chaminda and Pushpa HERATH, 2019. *Hands-On Functional test automation: With Visual Studio 2017 and Selenium*. B.m.: Apress.
7. PAJANKAR, Ashwin, 2017. *Python unit Test Automation: Practical Techniques for Python Developers and Testers*. B.m.: Apress.

Vedoucí bakalářské práce: **Ing. Alžběta Turečková**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

V této práci jsou prostudovány dostupné metody a nástroje vhodné pro automatizované hodnocení Python kódu v kontextu vzdělávání. Dále jsou zde zanalyzovány výhody a nevýhody automatizovaného hodnocení oproti manuálnímu přístupu.

Na základě této analýzy je navrženo vlastní řešení pro automatizované hodnocení Python kódu a několik příkladů demonstrujících použití navrženého řešení v praxi.

Klíčová slova: Python, LupyterLab, Nbgrader, Automatizované hodnocení, Edukace v informatice

ABSTRACT

This thesis focuses on available methods and tools suitable for automated evaluation of Python code in the context of education. Furthermore, the thesis analyzes advantages and disadvantages of automated evaluation over manual approaches.

Based on this analysis, a custom solution for automated evaluation of Python code is proposed and several examples demonstrating the use of the proposed solution in practice are presented.

Keywords: Python, LupyterLab, Nbgrader, Automated Assessment, Computer Science Education

Chtěl bych vyjádřit své upřímné poděkování Ing. Alžbětě Turečkové, Ph.D., za její profesionální vedení, cenné rady a podporu během psaní mé bakalářské práce. Čas, který mi věnovala mi byl neocenitelnou pomocí.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

S ohledem na důležitost počítačového vzdělávání v době stále rostoucí potřeby digitálního prostředí a popularitu Pythonu jako vstupního programovacího jazyka, nabývá automatizace testování kódu v kontextu vzdělávání stále většího významu. Účelem testování je nejen ověřit znalosti a dovednosti studentů, ale také poskytnout jim zpětnou vazbu a podnět k dalšímu rozvoji. Současně roste i potřeba efektivního a přístupného výukového prostředí pro výuku algoritmizace, které by bylo schopno oslovit a zapojit studenty různých úrovní znalostí a dovedností.

Cílem této práce je představit a prozkoumat možnosti využití programovacího jazyka Python v kontextu vzdělávání, se zaměřením na analýzu existujících metod a nástrojů pro testování kódu a navrhnout vlastní řešení pro automatizaci testování kódu studentů napsaného v tomto programovacím Jazyce.

I. TEORETICKÁ ČÁST

1 PYTHON

Python je vysokoúrovňový, open source, univerzální programovací jazyk, vhodný k vytváření všemožných nástrojů a aplikací od doporučujících algoritmů až po software ovládající samořiditelná auta. Trvale se řadí mezi nejoblíbenější programovací jazyky a získal široké uplatnění v komunitě strojového učení. [1, 2]

1.1 The Zen of Python

Základní filosofie Pythonu nese název „The Zen of Python“. Jedná se o jakýsi soubor devatenácti krátkých zásad sloužících jako základní kámen osvědčených programátorských návyků a postupů pro efektivní psaní kódu v jazyce Python. The Zen of Python zdůrazňuje důležitost čitelnosti, jednoduchosti, přehlednosti, konzistence a správných postupů.

Některými z bodů Zenu jsou:

Krása je lepší než ošklivost.

Jednoduché je lepší než složité.

Komplexní je lepší než komplikované.

Řídké je lepší než husté.

Na čitelnosti záleží.

Ve zkratce lze hlavní zásady Zenu shrnout následovně:

Pište snadno čitelný a srozumitelný kód se zaměřením především na jednoduchost, přehlednost a jednoznačnost. Chyby zpracovávejte elegantně a poskytujte informativní chybové zprávy. Dodržujte zavedené osvědčené postupy a konvence a usilujte o konzistentnost a přehlednost kódu.

Vývojáři Pythonu však sami tyto zásady občas porušují a dočkaly se kritiky za to, že jazyk zbytečně nafukují, zhoršují jeho čitelnost a zvyšují nároky na jeho učení. Kritika se týká např. přidání příkazu match/case nebo and operátorů pro sjednocení slovníků.

Navzdory tomu poskytují tyto principy užitečný rámec pro vzdělávání a výuku programování. Např. důraz na čitelnost, jednoduchost a explicitnost kódu může vést k lepšímu

porozumění studentům, kteří se teprve seznamují se základy programováním. Tyto principy mohou také pomoci studentům pochopit důležitost strukturovaného a systematického přístupu k řešení problémů a mohou ve studentech podněcovat potřebu k hledání elegantních a efektivních řešení.

Aktivní využívání těchto principů může pomoci nejen při osvojování konkrétních programovacích dovedností, ale také při rozvoji obecných myšlenkových postupů a přístupů k řešení problémů. [3, 4, 5]

1.2 K čemu se Python používá?

Python je mimořádně užitečným nástrojem nejen v oblasti programování, ale i ve vzdělávání. Jeho flexibilita a snadná syntaxe umožňují efektivní využití při výuce různých principů algoritmizace a vývoje softwaru. Python se běžně požívá při tvorbě webových stránek a aplikací, automatizaci úloh, skriptování, analýze a vizualizaci dat, ke strojovému učení, testování softwaru atd. Vzhledem k tomu, že se dá Python poměrně snadno naučit, osvojilo si jej, k vykonávání různých každodenních úkonů, i mnoho neprogramátorů. Jako např. účetní nebo vědci. [2]

1.2.1 Analýza dat a strojové učení

Jazyk Python umožňuje datovým analytikům a dalším odborníkům používat tento jazyk k provádění složitých statistických výpočtů, vytváření vizualizací dat, sestavování algoritmů strojového učení, manipulaci s daty a jejich analýze a k provádění dalších úkolů souvisejících s daty. V jazyce Python lze vytvářet širokou škálu různých vizualizací dat. Např. histogramy nebo nejrůznější typy grafů. Python má také řadu knihoven, které programátorům umožňují rychleji a efektivněji psát programy pro analýzu dat a strojové učení. Mezi tyto knihovny patří např. TensorFlow nebo Keras. [2]

1.2.2 Vývoj webu

Python se často používá k vývoji back endu webových stránek a aplikací. Úloha Pythonu ve vývoji webu může zahrnovat odesílání dat na servery i ze serverů, zpracování dat, komunikaci s databázemi, URL routování a zajištění bezpečnosti.

Python nabízí několik frameworků pro vývoj webových aplikací. Mezi běžně používané patří např. Django a Flask. [2]

1.2.3 Automatizace a skriptování

Pokud se stává, že je nějaký úkol prováděn opakovaně, může být velmi efektivní ho s pomocí Pythonu zautomatizovat. Tvorba kódu, sloužícího k automatizaci těchto procesů se nazývá skriptování. Ve světě kódování lze automatizaci využít ke kontrole chyb v souborech, k přejmenování či konverzi souborů, provádění různých matematických operací, odstraňování duplicit v datech atd. [2]

1.2.4 Testování software

Při vývoji software může být Python užitečným pomocníkem v činnostech jako je kontrola sestavení, bug tracking a testování. Pomocí jazyka Python mohou vývojáři softwaru automatizovat testování nových produktů nebo funkcí. V kontextu vzdělávání může mít Python zásadní roli při automatizovaném hodnocení výstupu studentů a testování jejich znalostí. Mezi nástroje Pythonu používané pro testování softwaru patří např. Pytest nebo Green. [2]

1.2.5 Každodenní úkony

Python není jen pro programátory a datové vědce. Naučit se ho může otevřít nové možnosti i pedagogům a studentům. Lze ho totiž uplatnit i pro zjednodušení a automatizaci každodenních úkonů ve vzdělávacím procesu, jako např. sledování vývoje studentů, sběr a analýza dat o jejich pokroku nebo poskytování zpětně vazby. Python může také umožnit zjednodušení některých úkonů v každodenním životě. Ať už se jedná o sledování cen na burze, zasílání připomínek nebo vyplňování online formulářů. [2]

1.3 Funkce Pythonu

Jedním z důležitých aspektů, které dělají z Pythonu ideální jazyk pro vzdělávací účely, je fakt, že používá automatickou správu paměti a dynamické typování, což znamená, že typ proměnné je určen až za běhu programu, a nikoli při jeho kompilaci.

Ke správě paměti používá garbage collector, jehož hlavním úkolem je sledovat objekty v paměti a odstraňovat ty, které již nejsou programem používány. Tím se zabraňuje úniku paměti a udržuje se její efektivní využití. Python také používá dynamické rozlišování jmen, které váže jména metod a proměnných za běhu programu.

Tyto vlastnosti Pythonu studentům umožňují soustředit se primárně na základní programovací koncepty a logiku, aniž by se museli hned zpočátku zabývat složitějšími tématy, jako je explicitní správa paměti nebo striktní typování.

Další užitečnou vlastností je podpora více programovacích paradigmat, včetně strukturovaného (zejména procedurálního), objektově orientovaného a funkcionálního programování, ale také mnoha dalších, která jsou podporována prostřednictvím jednotlivých rozšíření. To umožňuje studentům nejen naučit se základní procedurální programování, ale také objektově orientované a mnohé další programovací techniky. To vše v rámci stejného jazyka. Tato flexibilita nejenže umožňuje širší pedagogický přístup k výuce programování, ale nabízí také cenný přehled o různých způsobech řešení problémů v softwarovém inženýrství. [1]

1.4 Syntaxe

Syntaxe programovacího jazyka Python je soubor pravidel, která určují, jak bude program zapsán a interpretován, a to jak běhovým systémem, tak i lidskými čtenáři. Jazyk Python má mnoho podobností s jazyky Perl, C a Java. Mezi těmito jazyky však existují určité rozdíly.

Jednou z největších předností jazyka Pythonu, která jej činí ideálním jazykem pro vzdělávání v oblasti programování, je právě jeho jednoduchá a přehledná syntaxe. Python byl navržen s důrazem na snadnou čitelnost kódu, což umožňuje programy snáze jak číst, tak i psát.

Tato přístupnost umožňuje studentům rychle se ponořit do základů programování a efektivněji uchopit pojmi jako jsou cykly, podmíněné větvení a funkce. A to vše bez nutnosti překonávat bariéru složité syntaxe, která může být přítomna v jiných jazycích.

Syntaxe Pythonu se snaží řídit zásadou, že v nejlepším případě by měl existovat pouze jeden zřejmý způsob, jak něco udělat. Jazyk obsahuje vestavěné datové typy a struktury, control flow mechanismy, first-class funkce a moduly pro lepší znovu použitelnost a organizaci kódu. Python často používá anglická klíčová slova tam, kde jiné jazyky používají interpunkci, což přispívá k jeho specifickému vizuálnímu uspořádání. Jazyk dále poskytuje robustní ošetření chyb pomocí výjimek a ve své standardní knihovně obsahuje debugger pro efektivní řešení problémů.

Díky syntaxi jazyka Python, která je navržena pro srozumitelnost a snadné používání, je Python oblíbenou volbou jak mezi začátečníky, tak mezi profesionály. [6, 7, 8]

1.4.1 Klíčová slova

Python má 33 klíčových nebo rezervovaných slov, které nelze použít jako identifikátory. Kromě toho má také 4 „měkká“ klíčová slova. Na rozdíl od běžných („tvrdých“) klíčových slov, jsou měkká klíčová slova vyhrazena pouze v omezeném kontextu, kdy by jejich

interpretace jakožto klíčových slov dávala syntaktický smysl. Toto rozlišení se provádí na úrovni parseru, nikoli při tokenizaci.

Tato měkká klíčová slova se tedy dají použít jako identifikátory i jinde. V Pythonu lze například definovat funkci nebo proměnnou s názvem match nebo case. [8]

Tabulka 1. Seznam tvrdých klíčových slov v Pythonu

Klíčové slovo	*Popis*
False	Logická hodnota, výsledek porovnávacích operací
None	Představuje nulovou hodnotu
True	Logická hodnota, výsledek porovnávacích operací
and	Logický operátor
as	Vytvoření aliasu
assert	Pro ladění
break	Přerušení cyklu
class	Definice třídy
continue	Pokračování do další iterace cyklu
def	Definice funkce
del	Odstranění objektu
elif	Používá se v podmíněných výrazech, stejný význam jako else if
else	Používá se v podmíněných výrazech
except	Používá se u výjimek, co dělat, když nastane výjimka
finally	Používá se u výjimek, blok kódu, který se vykoná ať dojde k výjimce nebo ne
for	Vytvoření cyklu for
from	Import určitých částí modulu
global	Deklarace globální proměnné
if	Vytvoření podmíněného výrazu
import	Import modulu
in	Pro zjištění, zda je hodnota v seznamu, tuplu apod.
is	Testuje, zda se dvě proměnné rovnají
lambda	Vytvoření anonymní funkce
nonlocal	Deklarace nelokální proměnné
not	Logický operátor
or	Logický operátor
pass	Instrukce, která nic nedělá (zástupný výraz pro budoucí kód)
raise	Vyvolání výjimky

return	Ukončení funkce a vrácení hodnoty
try	Vytvoření try...except výrazu
while	Vytvoření cyklu while
with	Ke zjednodušení zpracování výjimek
yield	K vrácení seznamu hodnot z generátoru

Tabulka 2. Seznam měkkých klíčových slov v Pythonu

Klíčové slovo	*Popis*
match	Porovnává hodnotu se vzory
case	Definuje vzor v rámci výrazu match
type	Získá datový typ objektu
_	Ignoruje nebo označuje nepoužitou proměnnou

1.4.2 Odsazení

V takzvaných „free form“ jazycích, které používají blokovou strukturu odvozenou z jazyka ALGOL, jsou bloky kódu odděleny závorkami ({ }) nebo klíčovými slovy. Pro psaní kódu v těchto jazycích většinou platí, že programátoři kód v bloku konvenčně odsazují, aby jej vizuálně oddělili od okolního kódu.

Python, na rozdíl od těchto jazyků, používá k označování běhu bloků odsazení místo interpunkce nebo klíčových slov. K tomuto účelu slouží tzv. bílé znaky. Tuto vlastnost si vypůjčil od svého předchůdce, jazyka ABC.

Ačkoli jsou jako formy odsazení akceptovány jak znaky mezery, tak tabulátoru a lze použít jejich libovolný násobek, je doporučeno používat mezery, a to v násobcích čísla 4.

Míchání mezer a tabulátorů na po sobě jdoucích řádcích není od Pythonu 3 povoleno, protože to může vést k vytváření velmi obtížně viditelných chyb. Mnoho textových editorů totiž vizuálně nerozlišuje mezery a tabulátory.

Důraz na odsazení a vizuální strukturu kódu přispívá k vývoji dobrých programovacích návyků, jakým je i udržování čistoty a organizace kódu. Získání těchto návyků je klíčové pro každého softwarového vývojáře. A vzhledem k tomu, že Python používá explicitní syntaxi a vyžaduje, aby byly bloky kódu správně odsazeny, naučí se studenti, jak psát nejen funkční, ale i dobře strukturovaný a čitelný kód. [8, 9]

Zde je krátký příklad ilustrující správné použití odsazení v Python kódu:

```python
# Správně:

# Zarovnáno s počátečním oddělovačem.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Přidání další úrovně odsazení (4 mezer), kvůli oddělení argumentů od zbytku.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

Obrázek 1. Správně použité odsazení

```python
# Špatně:

# Argumenty na prvním řádku jsou zakázány, pokud není použito svislé zarovnání.
foo = long_function_name(var_one, var_two,
    var_three, var_four)

# Je nutné další odssazení, protože použité odsazení není dostatečně rozlišitelné.
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)
```

Obrázek 2. Špatně použité odsazení

1.4.3 Datové struktury

Protože Python je jazyk používající dynamickou typovou kontrolu, nesou informaci o typu hodnoty Pythonu, nikoli proměnné. Všechny proměnné v jazyce Python obsahují odkazy na objekty a tyto odkazy jsou předávány funkcím. Někteří lidé, včetně samotného tvůrce Pythonu Guida van Rossuma, nazývají toto schéma předávání parametrů jako volání pomocí reference objektu. Reference na objekt znamená jméno a předávaná reference je tzv. alias, tj. kopie reference na stejný objekt, stejně jako v jazyce C/C++. Hodnota objektu může být ve volané funkci pomocí aliasu změněna.

```
>>> my_list = ['a', 'b', 'c']
>>> def my_function(mlist):
...     mlist.append('d')
...     print(mlist)
...
>>> my_function(my_list)
['a', 'b', 'c', 'd']
>>> my_list
['a', 'b', 'c', 'd']
```

Obrázek 3. Volání funkce se změnou objektu pomocí aliasu

Funkce my_function mění hodnotu my_list pomocí formálního argumentu mlist, který je aliasem my_list. Jakýkoli pokus o operaci (přiřazení nové reference na objekt) na samotném aliasu však nebude mít žádný vliv na původní objekt.

```
>>> my_list = ['a', 'b', 'c']
>>> def my_function(mlist):
...     mlist = mlist + ['d']
...     print(mlist)
...
>>> my_function(my_list)
['a', 'b', 'c', 'd']
>>> my_list
['a', 'b', 'c']
```

Obrázek 4. Volání funkce beze změny objektu

V jazyce Python jsou všechny non-innermost-local a nedeklarované globální přístupné názvy aliasy.

Mezi dynamicky typovanými jazyky je Python středně typově kontrolován. Implicitní konverze je definována pro číselné typy, stejně jako pro logické. Takže lze např. platně násobit komplexní číslo celým číslem bez explicitního přetypování. Neexistuje zde však žádná implicitní konverze mezi čísly a řetězci. Takže u matematické funkce, která očekává číslo, nelze jako platný argument předat např. řetězec.

Python má širokou škálu základních datových typů. Vedle běžné aritmetiky celých čísel a čísel s pohyblivou desetinnou čárkou transparentně podporuje aritmetiku s libovolnou přesností a komplexní čísla.

Python také podporuje širokou škálu operací s řetězci. Ty jsou v jazyce Python neměnné, takže operace s nimi (např. záměna znaků), která by v jiném programovacím jazyce mohla změnit řetězec na místě, vrátí v jazyce Python řetězec nový.

Jedním z velmi užitečných aspektů jazyka Python je koncept kolekcí, resp. kontejnerů. Obecně platí, že kolekce je objekt, obsahující jiné objekty, na které se dá snadno odkazovat, nebo je lze snadno indexovat. Kolekce v Pythonu mají dvě základní podoby: sekvence a mapy.

Uspořádanými sekvenčními typy jsou list, tuple a string. Všechny sekvence jsou v Pythonu indexovány od 0 až po length - 1 a všechny, kromě řetězců, mohou obsahovat libovolný typ objektu. V jedné sekvenci se může nacházet i více různých typů. Listy jsou dynamická pole a jako takovým do nich lze prvky libovolně vkládat, mazat, upravovat a řadit. Zatímco stringy a tuply jsou neměnné. Což z nich dělá ideální kandidáty pro klíče slovníků (viz níže).

Mapy jsou naopak často neuspořádané typy implementované ve formě slovníků, které „mapují“ množinu neměnných klíčů na odpovídající prvky. Lze tedy např. definovat slovník, který má řetězec „toast“ mapovaný na celé číslo 42 nebo naopak. Klíče ve slovníku musí být neměnného typu (např. celé číslo nebo řetězec), protože jsou implementovány prostřednictvím hashovací funkce. To umožňuje mnohem rychlejší vyhledávání za cenu využití části paměti k uložení neměnných klíčů.

Slovníky jsou ústředním prvkem vnitřního systému jazyka Python. Tvoří totiž jádro všech objektů a tříd. Mapování mezi názvy proměnných a hodnotami, na které tyto názvy odkazují, jsou uloženy jako slovníky. Protože jsou tyto slovníky přímo přístupné, prostřednictvím atributu __dict__ objektu, je metaprogramování v Pythonu přímočarý a přirozený proces.

Python také poskytuje rozsáhlé schopnosti manipulace s kolekcemi, jako je vestavěný containment checking a obecný iterační protokol.

V jazyce Python je vše, včetně tříd, objektem. Třídy jako objekty mají svou třídu, která se nazývá metatřída. Python také podporuje vícenásobnou dědičnost a mixiny. Jazyk dále podporuje rozsáhlou introspekci typů a tříd. Typy lze číst a porovnávat. Atributy objektu lze získat jako slovník.

Operátory lze v Pythonu přetěžovat definováním speciálních členských funkcí. Např. definování metody s názvem __add__ ve třídě umožňuje používat operátor + na objektech této třídy. [10]

1.5 Výhody a nevýhody

Každý programovací jazyk má své přednosti i svá omezení, které lze označit jako výhody a nevýhody. V této kapitole budou postupně popsány některé z nich.

1.5.1 Výhody

Python nabízí hned několik výhod, které jej činí vynikajícím jazykem jak pro vzdělávací účely, tak pro použití v kontextu automatizace hodnocení studentů.

Výhody, jako je jeho snadná učení, široká podpora knihoven a schopnost adaptovat se různým vývojovým potřebám, z něho dělají silný nástroj pro zavádění principů programování a algoritmizace. Jednoduchá syntaxe a čitelnost kódu usnadňuje studentům rychlé pochopení základů programování, podporuje učební proces.

Mezi hlavní výhody patří:

Snadné učení, čtení a porozumění: Syntaxe Pythonu je jednoduchá a přehledná, což usnadňuje učení se tohoto jazyka jak začátečníkům, tak zkušenějším programátorům. Dobrá čitelnost kódu je jedním z hlavních rysů Pythonu, což zjednodušuje údržbu a spolupráci na projektech.

Všestrannost a open-source: Python je univerzální programovací jazyk, který lze použít pro širokou škálu aplikací, včetně vývoje webových aplikací, analýzy dat, strojového učení, automatizace, vědeckého výzkumu a mnoha dalších. Je také distribuován pod licencí open source. Což znamená, že ho lze používat zcela zdarma a komunita může přispívat k jeho vývoji.

Zlepšuje produktivitu: Python nabízí jednoduché a efektivní nástroje pro řešení problémů a díky svým vlastnostem, jako je dynamické typování a automatická správa paměti, umožňuje rychlejší vývoj softwaru.

Podpora rozsáhlých knihoven: Python má rozsáhlou knihovnu modulů a balíčků, které pokrývají téměř každou oblast vývoje softwaru. Tato paleta knihoven umožňuje vývojářům vytvářet aplikace rychleji a efektivněji, díky využívání existujícího kódu a funkcionalit.

Silná komunita: Python má velkou a aktivní komunitu vývojářů, kteří poskytují cenné rady, sdílejí znalosti a vytvářejí další užitečné nástroje a knihovny. Tato komunita významným způsobem přispívá k rozvoji jazyka a udržuje jeho popularitu.

Interpretovaný jazyk: Python je interpretovaný jazyk, což znamená, že kód je spouštěn řádek po řádku. Tento přístup usnadňuje ladění a testování aplikací.

Integrace s testovacími nástroji: Python umožňuje snadnou integraci s nástroji pro testování a automatizaci, jako jsou Pytest, Unittest nebo Jupyter Notebook, které mohou být použity k vytváření interaktivních výukových materiálů a automatizovaného hodnocení úkolů. To nejen zvyšuje efektivitu hodnocení, ale také poskytuje studentům okamžitou zpětnou vazbu k jejich práci.

1.5.2 Nevýhody

Některé vlastnosti Pythonu mohou být považovány za omezující, což může vést k určitým kompromisům při tvorbě složitých systémů. Mezi tyto vlastnosti je možné zahrnout následující:

Omezená podpora vláken: Omezená podpora více vláknového programování v Pythonu je důsledkem mechanismu „Global Interpreter Lock“ (GIL). Ten totiž v danou chvíli umožňuje provádět svůj kód pouze jednomu vláknu.

Nízká efektivita paměti: Python má oproti jiným jazykům tendenci být paměťově náročnější, a to zejména při zpracování velkých objemů dat. To může vést k problémům s výkonem v paměťově omezených prostředích.

Slabá podpora mobilních výpočtů: Python není ideální pro vývoj mobilních aplikací. Ve srovnání s jinými jazyky optimalizovanými pro mobilní platformy má totiž slabí výpočetní výkon a omezenou podporu pro mobilní výpočty.

Chyby za běhu: Jelikož je Python dynamicky typovaným a interpretovaným jazykem, nemá striktní statickou typovou kontrolu. To může vést k chybám za běhu, které by bylo možné, v případě statického typování, odhalit už při kompilaci.

Omezená syntaxe: Syntaxe Pythonu je relativně jednoduchá a snadno čitelná, což je jedna z jeho největších výhod. Nicméně některé pokročilé programátorské konstrukce, jako např. lambda funkce, mohou být ve srovnání s jinými programovacími jazyky omezené, nebo zcela nepřítomné. Tento fakt může být překážkou pro pokročilé uživatele.

Po zhodnocení těchto kladů a záporů lze říci, že přínosy Pythonu pro vzdělávací sektor a jeho použití pro automatizované hodnocení výstupů studentů převažují nad jeho omezeními. Python poskytuje mocný základ pro výuku softwarového inženýrství a otevírá dveře k praktickému uplatnění získaných dovedností. [7, 11]

2 JUPYTERLAB

Jupyter je rozsáhlý a funkčně bohatý zastřešující projekt, který zahrnuje mnoho různých softwarových nástrojů, včetně webové aplikace pro tvorbu a editaci zápisníků, známé jako Jupyter Notebook. Projekt Jupyter a jeho podprojekty se soustředí na poskytování standardů a nástrojů pro interaktivní výpočty pomocí výpočetních notebooků.

Notebook je sdílený dokument, který kombinuje počítačový kód, popisy v běžném jazyce, data, různé vizualizace, jako jsou 3D modely, grafy a obrázky, a interaktivní ovládací prvky. Notebook spolu s editorem, jako je např. JupyterLab, poskytuje rychlé interaktivní prostředí pro vysvětlování kódu, zkoumání a vizualizaci dat a sdílení nápadů s ostatními.

JupyterLab poskytuje integrované a rozšiřitelné prostředí, kde je možné nejen upravovat notebooky, ale také pracovat s konzolami a textovými editory v jednotném rozhraní. Toto prostředí je ideální pro rozsáhlejší projekty, které vyžadují manipulaci s více dokumenty a nástroji současně. [12, 13, 14]

2.1 Hlavní funkce webové aplikace JupyterLab

JupyterLab poskytuje uživatelsky přívětivé rozhraní s podporou funkcí jako automatické zvýrazňování syntaxe, odsazování a doplňování tabulátorů, což umožňuje snadnou úpravu textu přímo v prohlížeči. Díky možnosti spouštět kód přímo z prohlížeče je JupyterLab ideálním prostředím pro rychlé experimentování a ladění kódu. Výsledky výpočtů jsou okamžitě propojeny s odpovídajícím kódem, což usnadňuje analyzování a interpretaci výsledků.

JupyterLab umožňuje zobrazovat výsledky výpočtů pomocí různých mediálních reprezentací, jako jsou HTML, LaTeX, PNG, SVG a další. To umožňuje uživatelům prezentovat výsledky svých analýz a vizualizací v přehledné a atraktivní formě.

V JupyterLabu je možné psát pomocí značkovacího jazyka komentáře ke kódu, vysvětlit jeho části a dokumentovat postup své práce. Je zde také možnost snadného začlenění matematického zápisu pomocí LaTeXu a nativního vykreslování matematických výrazů pomocí MathJax. Tato funkce je obzvláště užitečná pro vědecké a technické aplikace, kde je často potřeba zobrazovat matematické rovnice a symboly. [14]

2.2 Notebook soubory

Notebook soubory obsahují vstupy a výstupy interaktivní relace a další doprovodný text, který popisuje nebo jinak doplňuje kód, ale sám není určen k provedení. Tímto způsobem

mohou soubory notebooků sloužit jako kompletní výpočetní záznam relace, v němž se prolíná spustitelný kód s vysvětlujícím textem, matematikou a bohatou reprezentací výsledných objektů.

Notebooky lze exportovat do řady statických formátů, včetně HTML, reStructuredText, LaTeX, PDF a prezentací, pomocí příkazu nbconvert.

Kromě toho lze jakýkoli soubor notebooku .ipynb dostupný z veřejné adresy URL sdílet prostřednictvím prohlížeče Jupyter Notebook Viewer. Tato služba načte dokument notebooku z adresy URL a vykreslí jej jako statickou webovou stránku. Výsledky tak lze sdílet s kolegy či studenty nebo jako veřejný příspěvek na blogu, aniž by ostatní uživatelé museli sami instalovat JupyterLab. [14]

2.3 Zabezpečení a soukromí

Vzhledem k tomu, že uživatelské rozhraní Jupyteru běží ve webovém prohlížeči, vyvstávají otázky ohledně soukromí a zabezpečení citlivých údajů. Pokud byl však Jupyter nainstalován podle standardních instalačních pokynů, běží ve skutečnosti na lokálním počítači, který funguje jako lokální server. Jupyter tedy standardně neposílá data nikam jinam, což je možné si díky jeho open source podstatě kdykoli ověřit.

Jupyter je však možné používat i na dálku. Lze se např. připojit k serveru univerzity nebo firmy, pokud to daná instituce umožňuje. A protože přístup k Jupyter serveru znamená i přístup k spouštění libovolného kódu, je důležité přístup k serveru určitým způsobem omezit. K tomuto účelu se používá ověřování na bázi tokenů, zabezpečení heslem nebo kombinace obou možností. [15]

3 TESTOVÁNÍ SOFTWARU

Testování softwaru je organizační proces v rámci vývoje softwaru, ve kterém jsou softwarové produkty ověřovány z hlediska správnosti, kvality a výkonnosti. Testování softwaru se používá k zajištění toho, že softwarový produkt odpovídá očekávaným požadavkům a obsahuje co nejmenší množství chyb.

Z hlediska vzdělávání je testování softwaru důležitou aktivitou, která podporuje jak proces učení studentů, tak vývoj jejich programovacích dovedností. Výuka programování vyžaduje nejen pochopení teoretických konceptů, ale také praktické ověření a validaci psaného kódu. Testování v tomto směru nabízí neocenitelnou zpětnou vazbu, která studentům umožňuje rychle identifikovat a opravit chyby ve svém kódu a zároveň posiluje jejich porozumění programovacím principům. [16, 17]

3.1 Manuální testování

Manuální testování je řízeno týmem nebo jednotlivcem, který zastává roli koncového uživatele. Ten se softwarem manuálně provádí nejrůznější operace a zjišťuje, zdali se software chová podle očekávání. Tato metoda je založena na lidském úsudku a interakci se softwarem.

Jednou možností manuálního testování je testování pomocí testovacích scénářů. V tomto případě provádí tester manuálně různé akce podle předem definovaných kroků a sleduje, zda se chování softwaru shoduje s očekáváním.

Druhou možností je testovat neplánovaně. U této možnosti tester naprosto spontánně zkoumá aplikaci a pokouší se identifikovat možné chyby a nedostatky. Tato možnost se používá k objevení nepředvídatelných problémů. [16]

3.2 Automatizované testování

Automatizované testování softwaru je složeno z mnoha různých nástrojů, které mají různé schopnosti. Od izolovaných kontrol správnosti kódu až po simulaci plně manuálního testování prováděného člověkem. Implementace automatizovaných testů vyžaduje pečlivé plánování a návrh. Správně navržené automatizované testy jsou však schopny provádět akce a kontrolovat aspekty softwaru s konzistentním výsledkem a mohou výrazně zvýšit kvalitu softwaru a urychlit kontrolu jeho kvality.

Existuje několik typů automatizovaných testů, které lze v rámci vývoje softwaru použít. Mezi tyto typy patří: jednotkové testy, integrační testy, funkční testy, systémové testy a

akceptační testy. Těmito typy se budeme podrobněji zabývat v kapitole 3.4 Úrovně softwarového testování. [16]

3.3 Srovnání manuálního a automatizovaného testování

Manuální testování je zpravidla flexibilnější, zvládá se snadno přizpůsobit měnícím se podmínkám a snadněji reagovat na nepředvídané situace. Tester může snadno upravit testovací postupy a scénáře podle aktuálních potřeb a okolností. Zatímco automatizované testy nabízejí konzistentní a opakovatelné výsledky. Jsou však méně flexibilní, co se týče reakcí na nečekané změny v softwaru. Úpravy automatizovaných testů mohou vyžadovat čas a úsilí, zejména je-li potřeba upravit testovací skripty nebo testovací frameworky.

Manuální testy mohou být časově velmi náročné, zejména pokud jsou testy často opakovány nebo pokud se jedná o rozsáhlejší testovací scénář. Opakování manuálních testů tedy vede ke zbytečné ztrátě času a zvyšování nákladů. Automatizované testy jsou rychlejší a umožňují opakování testů s minimálním lidským zásahem. To může vést k významné úspoře času a nákladů, zejména při opakovaném spouštění testovacích sestav.

Manuální testování závisí na lidské intuici a subjektivním úsudku, což může u různých testerů vést k různým výsledkům. U rozsáhlejších a opakujících se testovacích postupů je tedy obtížné dosáhnout konzistentních výsledků. Automatizované testy naproti tomu poskytují konzistentní výsledky při každém provedení a umožňují snadnou kontrolu pokrytí testování. Díky tomu je zajištěno, že při testování nedojde k opomenutí některé z částí kódu.

Jednotlivé rozdíly mezi manuálním a automatizovaným testováním je třeba pečlivě zvážit v kontextu konkrétních požadavků, časových omezení a charakteristik testovaného softwaru. Ideálním řešením pro dosažení optimálního výsledku může být kombinace obou metod. [18, 19, 20]

3.4 Úrovně softwarového testování

V rámci vývojového procesu existuje několik základních úrovní softwarového testování. Každá zkoumá funkčnost softwaru z jiného hlediska. V této části bude postupně popsána každá z nich a její praktické použití v kontextu vzdělávání.

3.4.1 Jednotkové testování

Základní úrovní testování softwaru je jednotkové testování. Tento typ testování je skvělým způsobem, jak ověřit správnost kódu. Jedná se o způsob kontroly správnosti vstupu a výstupu

pro jednotlivé části kódu izolovaně od zbytku aplikace. Měřenými jednotkami jsou v tomto případě samostatné funkce nebo metody.

Během tohoto typu testování jsou produkční funkce kódu prováděny v testovacím prostředí se simulovanými vstupy. Výstup funkce se pak porovnává s očekávaným výstupem pro daný vstup. Pokud výstup odpovídá očekávanému, test proběhl úspěšně. V opačném případě se jedná o neúspěch.

Jednotkové testování umožňuje studentům ověřit funkčnost jednotlivých komponent kódu. Je základním stavebním kamenem pro výuku dobrých testovacích praktik a pomáhá studentům osvojit si základy izolovaného testování. [16]

Jednotkový test funkce add(x, y), která vrací součet parametrů x a y, by mohl vypadat např. takto:

```python
import unittest

def secti(x, y):
    return x + y

class TestSectiFunkce(unittest.TestCase):

    def test_secti_kladna_cisla(self):
        self.assertEqual(secti(2, 5), 7)

    def test_secti_zaporna_cisla(self):
        self.assertEqual(secti(-2, -5), -7)

    def test_secti_smisena_cisla(self):
        self.assertEqual(secti(2, -5), -3)

if __name__ == '__main__':
    unittest.main()
```

Obrázek 5. Příklad jednotkového testu

3.4.2 Integrační testování

Cílem integračních testů je ověřit interakce mezi jednotlivými komponentami v rámci aplikace, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů. Pokud testovací případ pokrývá více než jednu jednotku, je považován za integrační. Při vývoji testovacího případu se tak může hranice mezi jednotkovým a integračním testem rychle prolomit. Často se totiž stává, že je vyvinut jednotkový test, který funguje proti

závislosti na kódu třetí strany. Samotná závislost však nemusí být testována a integrace s ní tedy bude pouze předstírána. Na tomto typu testování se většinou nepodílejí testeři, ale přímo vývojáři.

Pro studenty je integrační testování skvělou příležitostí ukázat si, jak moduly softwaru spolupracují a pochopit vztahy mezi různými částmi systému. Dále se při tomto typu testování mohou studenti naučit identifikovat a řešit problémy způsobené integrací. [16, 17, 21]

3.4.3 Funkční testování

Testovací případy, které simulují plný uživatelský zážitek, se nazývají funkční testy nebo někdy též end-to-end testy. Tyto testy používají nástroje simulující reálné chování lidských uživatelů.

Funkční testování je proces ověřování funkcionality softwarového produktu z hlediska splnění požadovaných specifikací a očekávaného chování. Hlavním cílem funkčních testů je zjišťovat, zda software provádí požadované úkoly a interakce tak, jak bylo navrženo v dokumentaci.

Tyto testy tedy ověřují správné fungování všech implementovaných funkcí se zaměřením na různé aspekty funkcionality, jako jsou uživatelská rozhraní, vstupy a výstupy, datové operace atd. Běžnými kroky v těchto testech jsou: kliknutí na tlačítko, čtení textu, odeslání formuláře atp. Další důležitou součástí funkčního testování je ověření, zda software řádně zpracovává chybové stavy a neplatné vstupy, a zda poskytuje odpovídající chybové zprávy a návody k řešení.

Tento druh testování pomáhá studentům lépe pochopit, jak software interaguje s uživateli a jaké výsledky by měla produkce skutečně přinést. [16, 21, 22]

3.4.4 Systémové testování

Během systémového testování je celý softwarový systém ověřován jako jeden velký funkční celek. Systémové testování zajišťuje, že softwarový produkt splňuje specifikace a požadavky, týkající se výkonu a spolehlivosti, stanovené ve fázi návrhu a analýzy. Tento typ testování obvykle probíhá ve finální fázi vývoje před uvedením produktu do provozu. Jeho cílem je identifikovat a odstranit potenciální chyby a nedostatky před nasazením softwaru do reálného prostředí.

Systémové testy zahrnují testování funkcionality, výkonnosti, bezpečnosti, spolehlivosti a kompatibility s dalšími systémy a prostředími. Simulují se zde různé kroky podle předem připravených scénářů, které mohou v praxi nastat. Tyto testy zpravidla probíhají v několika kolech. Nalezené chyby jsou postupně opravovány a tyto opravy jsou poté v dalších kolech opět testovány.

Systémové testování se objevuje prakticky v každém testovacím procesu. Vynecháním této testovací úrovně by byla výrazně ohroženo bezporuchovost výsledného produktu a celé testování softwaru by nemělo zdaleka takový význam.

Tato fáze umožňuje studentům vidět, jak různé komponenty aplikace fungují společně pod zátěží v reálných provozních podmínkách. [21]

3.4.5 Akceptační testování

Pokud již proběhly všechny předchozí úrovně testů bez větších nedostatků, nastal čas k provedení akceptačních testů.

Cílem akceptačního testování je zjistit, zda je software připraven k uvedení do provozu, a zda vyhovuje potřebám a požadavkům koncových uživatelů. V edukačním kontextu to může znamenat, že projekt je připraven k odevzdání a zhodnocení.

Tento typ testování se provádí z pohledu uživatele (vyučujícího) a zaměřuje se na to, zda softwarový produkt plní jeho zamýšlené funkce a poskytuje očekávané hodnoty. Tato úroveň testování je většinou prováděna přímo v reálném prostředí.

Během akceptačního testování jsou testovány různé scénáře použití a prováděny testy na základě reálných podmínek, aby bylo zajištěno, že software funguje v souladu s očekáváním uživatele (vyučujícího). V případě nalezení nedostatků nebo nesrovnalostí během akceptačního testování jsou tyto problémy dokumentovány a předány vývojovému týmu (studentům) k opravě. [21]

3.5 Význam automatizovaného testování pro hodnocení studentů

Používání automatizovaných testů pro hodnocení studentů umožňuje učitelům objektivně měřit postup studentů a efektivně poskytovat zpětnou vazbu. Tento přístup podporuje konzistentní a spravedlivé hodnocení, zatímco poskytuje studentům cenné informace o jejich výkonu a oblastech, které potřebují zlepšit. Automatizované testování ve vzdělávacím

prostředí také napomáhá k vyšší transparentnosti ve výsledcích a usnadňuje výměnu nejlepších praktik mezi studenty a výukovými institucemi.

4 NÁSTROJE PRO AUTOMATIZACI TESTOVÁNÍ PYTHON KÓDU

Vývoj software a vzdělávání v oblasti programování vyžadují použití efektivních nástrojů pro testování kódu. Tyto nástroje nejen zvyšují efektivitu a přesnost testování, ale také poskytují studentům praktické zkušenosti s nástroji používanými v průmyslu. Tato kapitola se zaměřuje na několik klíčových nástrojů, které jsou široce používány pro testování Python kódu.

4.1 Unittest

Modul unittest je framework, jehož primárním cílem je usnadnit práci při testování kódu. Jeho funkce je založena na některých důležitých objektově orientovaných konceptech. Proto je k jeho pochopení potřeba porozumět základům tříd a metod v jazyce Python.

Testovací případ je zde považován za základní testovací jednotku a je reprezentován třídou TestCase. Mezi mnoha nástroji, které unittest poskytuje a které umožňují testovat kód, je tato třída jednou z nejdůležitějších. Používá se jako základní třída pro vytváření vlastních testovacích případů, které umožňují spouštět více testů najednou. [23, 24]

4.1.1 assert

Assert je výrazem vestavěným v programovacím jazyce Python, který slouží, jak jeho název napovídá, k potvrzení, zda je daná podmínka pravdivá, či nikoliv. Pokud je podmínka pravdivá, nic se nestane a kód nerušeně pokračuje v provádění svých instrukcí. V opačném případě je vyvolána chyba. Ačkoli se na první pohled může zdát, že se jedná o stejnou funkci, jakou má try a except, jedná se o zcela odlišné příkazy. Assert by neměl být používán pro ošetření chyb ale pouze k účelu ladění a testování. [23, 24]

4.1.2 Třída TestCase

Třída TestCase poskytuje několik vlastních metod assert, které fungují stejně jako příkaz assert, ale pouze pro specifické typy tvrzení.

Např. assertEqual vezme dva prvky a testuje, zda se navzájem rovnají, zatímco assertNotEqual testuje, zda se prvky liší. Metoda assertTrue zase bere jeden prvek a testuje, zda je pravdivý, zatímco assertFalse testuje, zda je nepravdivý.

Zde je tabulka nejčastěji používaných assert metod ve třídě TestCase podle oficiální dokumentace unittestu.

Tabulka 3. Nejčastěji používané assert metody v unittestu

Method	*Checks that*
assertEqual(a, b)	`a == b`
assertNotEqual(a, b)	`a != b`
assertTrue(x)	`bool(x) is True`
assertFalse(x)	`bool(x) is False`
assertIs(a, b)	`a is b`
assertIsNot(a, b)	`a is not b`
assertIsNone(x)	`x is None`
assertIsNotNone(x)	`x is not None`
assertIn(a, b)	`a in b`
assertNotIn(a, b)	`a not in b`
assertIsInstance(a, b)	`isinstance(a, b)`
assertNotIsInstance(a, b)	`not isinstance(a, b)`

Je důležité zmínit, že všechny assert metody ve třídě TestCase přijímají také argument msg, který se používá jako chybová zpráva v případě selhání testu. [23, 24]

4.2 Doctest

Doctest je modul v jazyce Python, který umožňuje psát testovací příklady přímo do dokumentace kódu pomocí tzv. docstringů a poskytuje tak efektivní způsob, jak kód zároveň dokumentovat i testovat.

Testovací příklady uvedené v docstringech jsou automaticky spouštěny a vyhodnocovány při spuštění modulu pomocí testovacího nástroje jako je například Pytest nebo přímo pomocí modulu Doctest. Doctest je snadno použitelný, protože nevyžaduje žádné dodatečné knihovny ani konfigurace.

Existuje několik běžných způsobů použití Doctestu.

Kontrola aktuálnosti dokumentace modulu tím, že ověří, zda všechny interaktivní příklady stále fungují tak, jak je zdokumentováno. Provedení regresního testování ověřením, zda interaktivní příklady z testovacího souboru nebo testovacího objektu fungují podle očekávání. Psaní výukové dokumentace k balíčku, hojně ilustrovanou vstupně-výstupními příklady. [25, 26]

4.3 Pytest

Pytest je rozsáhlý a oblíbený framework pro testování v jazyce Python. Je proslulí svou jednoduchou syntaxí, rozsáhlou podporou, flexibilitou a funkčně bohatým ekosystémem založeném na zásuvných modulech. Umožňuje psaní krátkých velmi snadno čitelných testů v podobě běžných funkcí v Pythonu a poskytuje funkce pro automatizaci a organizaci testovacích scénářů. Lze jej škálovat tak, aby podporoval komplexní funkční testování aplikací a knihoven.

Poskytuje také mechanismus fixture pro inicializaci a sdílení dat mezi testy. Tento mechanismus usnadňuje přípravu prostředí pro testování, zlepšuje modularitu testovacích scénářů a snižuje duplicitu dat. [27, 28]

4.4 CI/CD pipeline

CI a CD jsou dvě zkratky často používané v moderních vývojových postupech a DevOps. CI je zkratka pro continuous integration, což je základní osvědčený postup DevOps, který umožňuje vývojářům pravidelně slučovat změny kódu do centrálního úložiště, kde se pak automaticky spouští sestavení a testy. CD však může znamenat buď continuous delivery, nebo continuous deployment.

Continuous integration, delivery a deployment jsou tři fáze automatizovaného procesu uvolňování softwaru. Během těchto fází se software dostane od pouhého nápadu až po dodání koncovému uživateli. [21, 22, 23]

4.4.1 Continuous integration

Continuous integration (CI) je postup automatizace integrace změn kódu od více přispěvatelů do jednoho softwarového projektu. Před integrací se používají automatizované nástroje, aby se potvrdila správnost nového kódu.

Jádrem procesu CI je systém správy verzí zdrojového kódu. Systém řízení verzí je doplněn také dalšími kontrolami, jako jsou automatizované testy kvality kódu, nástroje pro kontrolu stylu syntaxe a další.

Vývojáři praktikující kontinuální integraci slučují své změny zpět do hlavní větve tak často, jak je to jen možné. Změny vývojářů se ověřují vytvořením sestavy a spuštěním automatizovaných testů proti této sestavě. Tímto způsobem se vyhnou problémům s integrací, které mohou nastat při slučování změn do hlavní větve až v den vydání.

Kontinuální integrace klade velký důraz na automatizaci testů, které kontrolují, zda není aplikace rozbitá, kdykoli jsou do hlavní větve integrovány nové revize. [21, 22, 23]

4.4.2 Continuous delivery

Continuous delivery (CDel) je rozšířením CI, protože po fázi sestavení automaticky nasazuje všechny změny kódu do testovacího, nebo produkčního prostředí. To znamená, že kromě automatizovaného testování má vývojář k dispozici i automatizovaný proces vydávání a aplikaci může kdykoli nasadit kliknutím na tlačítko.

Teoreticky se díky kontinuálnímu dodávání může vývojář rozhodnout, zda bude aplikaci vydávat denně, týdně, jednou za čtrnáct dní nebo podle toho, co vyhovuje jeho obchodním požadavkům. Pokud by však chtěl skutečně naplno využívat výhod které mu CDel přináší, měl by nasazovat do produkce co nejdříve, aby měl jistotu, že vydává malé sady, které lze v případě problému snadno opravit. [22, 23]

4.4.3 Continuous deployment

Continuous deployment (CDep) je ještě o krok dál než CDel. Při tomto postupu je každá změna, která projde všemi fázemi produkčního potrubí, uvolněna přímo zákazníkům. V tomto procesu není přítomen žádný lidský zásah a nasazení nové změny do produkce zabrání pouze neúspěšný test.

CDep je vynikající způsob, jak urychlit smyčku zpětné vazby se zákazníky a zbavit tým tlaku spojeným s dnem vydání. Vývojáři se mohou plně soustředit na tvorbu softwaru a sledovat, jak jde jejich práce do provozu jen několik minut poté, co na ní dokončili práci. [22, 23]

4.4.4 Význam CI

Absence CI vyžaduje manuální koordinaci mezi vývojáři, což může vést k pomalému vydávání kódu a vyšší míře selhání. Bez robustní CI pipeline může docházet k odpojení vývojářského týmu od zbytku organizace, což komplikuje komunikaci a odhad doby dodání požadavků.

CI pomáhá zvyšovat výkon vývojářských týmů tím, že jim umožňuje pracovat na funkcích nezávisle a současně s dalšími týmy. Když jsou pak tyto funkce připraveny k přidání do konečného produktu, mohou tak učinit pohotově a nezávisle.

CI se obvykle používá spolu s agilním pracovním postupem vývoje softwaru. Organizace sestaví seznam úkolů, které tvoří plán produktu. Tyto úkoly jsou pak rozděleny mezi členy týmu softwarového inženýrství, kteří je mají dodat. Použití CI umožňuje, aby tyto úkoly vývoje softwaru byly vyvíjeny nezávisle a paralelně mezi přidělenými vývojáři. Jakmile je jeden z těchto úkolů dokončen, vývojář zavede tuto novou práci do systému CI, aby byla integrována do zbytku projektu. [21]

4.4.5 Přednosti a úskalí CI

Kontinuální integrace je základním aspektem DevOps a vysoce výkonných softwarových týmů. Přínosy CI se však neomezují pouze na vývojářský tým, ale jsou velmi přínosné pro celou organizaci. CI umožňuje lepší transparentnost a přehled o procesu vývoje a dodávání softwaru. Tyto přínosy umožňují zbytku organizace lépe plánovat a realizovat strategie pro uvedení na trh. Zde jsou některé z organizačních přínosů CI.

Škálování: CI umožňuje organizacím škálovat velikost vývojářského týmu, velikost kódové základny a infrastruktury. Minimalizací byrokracie při integraci kódu a komunikační režie pomáhá CI budovat DevOps a agilní pracovní postupy. Umožňuje každému členovi týmu mít na povel konkrétní změnu kódu od začátku její tvorby až po vydání. CI umožňuje škálování tím, že odstraňuje veškeré organizační závislosti mezi vývojem jednotlivých funkcí. Vývojáři nyní mohou pracovat na jednotlivých funkcích izolovaně a mají jistotu, že jejich kód bude bezproblémově integrován se zbytkem kódové základny.

Zlepšení zpětnovazební smyčky: Dalším silně kladným vedlejším efektem CI je rychlejší zpětná vazba na obchodní rozhodnutí. Produktové týmy mohou díky optimalizované platformě CI rychleji testovat nápady a iterovat návrhy produktů. Změny lze rychle prosazovat a měřit jejich úspěšnost. Chyby nebo jiné problémy lze rychle řešit a opravovat.

Zlepšení komunikace: CI zlepšuje celkovou komunikaci, což umožňuje lepší spolupráci mezi vývojem a provozem. Zavedením pracovních postupů pull requestů navázaných na CI, sdílejí vývojáři pasivně své znalosti s ostatními účastníky vývojového procesu. Požadavky pull umožňují vývojářům sledovat a komentovat kód ostatních členů týmu. Vývojáři si nyní mohou prohlížet větve funkcí a spolupracovat na nich s ostatními vývojáři. Efektivní CI pipeline s vysoce spolehlivým automatickým pokrytím testů ochrání před regresemi a zajistí, že nové funkce odpovídají specifikaci. Přínosy CI zdaleka převažují nad případnými problémy při zavádění. Přesto je důležité uvědomit si problémy CI. Skutečné problémy CI vznikají při přechodu projektu z režimu bez CI na CI. Většina moderních softwarových projektů

si osvojí CI již v raných počátečních fázích a zmírní tak problémy spojené s pozdějším osvojením.

Přijetí a instalace: Problémy kontinuální integrace se týkají především přijetí týmem a počáteční technické instalace. Pokud tým v současné době nemá zavedené řešení CI, může vyžadovat určité úsilí, aby si ho vybral a začal používat. Při instalaci CI pipeline je tedy třeba brát ohledy na stávající technickou infrastrukturu.

Křivka technologického učení: Funkce CI je spojena se seznamem podpůrných technologií, které mohou pro tým představovat investici do křivky učení. Těmito technologiemi jsou systémy pro správu verzí, hostingová infrastruktura a orchestrační technologie. [21]

4.5 nbgrader

nbgrader je rozšíření pro JupyterLab, navrženo speciálně pro vzdělávací prostředí. Umožňuje vyučujícím efektivně vytvářet, distribuovat a automaticky hodnotit programovací úkoly. Nbgrader se zaměřuje na snížení administrativní zátěže spojené s průběžným hodnocením studentů a na zefektivnění celého výukového procesu.

4.5.1 Základní funkce

Automatické hodnocení: Nbgrader poskytuje nástroje pro definování automatických testů, které ověřují správnost řešení úkolů odevzdaných studenty. Tento přístup umožňuje objektivní a konzistentní hodnocení studentů.

Správa úkolů: Umožňuje vyučujícím snadno vytvářet úkoly s jasně definovanými kritérii hodnocení a distribuovat je studentům. Po termínu odevzdání může vyučující úkoly jednoduše shromáždit a vyhodnotit.

Zpětná vazba: Vyučující může generovat podrobné zpětné vazby, které pomáhají studentům pochopit jejich chyby a podpořit tak jejich učební proces.

4.5.2 Adresářová struktura

Následující sekce popisuje základní adresáře nbgraderu a jejich účel.

source: Obsahuje zdrojové Jupyter notebooky s úkoly, které připravuje lektor.

release: Sem jsou umísťovány zpracované notebooky určeny pro distribuci studentům.

submitted: Do tohoto adresáře se umisťují studenty odevzdané notebooky.

autograded: Adresář pro automaticky ohodnocené úkoly.

feedback: Adresář se soubory obsahující zpětnou vazbu pro studenty.

[29]

II. PRAKTICKÁ ČÁST

5 PŘÍPRAVA PROSTŘEDÍ

Jedním z cílů praktické části této práce je ukázat proces přípravy a konfigurace softwarového prostředí nezbytného pro efektivní výuku programování v jazyce Python s využitím JupyterLab a nbgrader. Tato kapitola podrobně popisuje kroky potřebné k instalaci a nastavení klíčových nástrojů, které umožní automatizaci hodnocení kódu studentů, a tím zefektivní jak výuku, tak i administrativu spojenou s průběžným hodnocením studijních výsledků.

Prostředí, ve kterém studenti a vyučující pracují, musí být pečlivě připraveno tak, aby podporovalo jak základní vývojové aktivity, tak specifické potřeby výukového procesu. To zahrnuje instalaci interpretu jazyka Python, vývojového prostředí JupyterLab a rozšíření nbgrader, které spolu vytvářejí robustní framework pro výuku programování a automatického hodnocení.

V následujících podkapitolách jsou popsány kroky potřebné pro přípravu tohoto prostředí, od získání a instalace Pythonu, přes nastavení JupyterLabu, až po implementaci nbgraderu. Tato příprava zahrnuje technické aspekty, jako jsou instalace software a konfigurace systémových cest, které jsou nezbytné pro bezproblémové fungování výukového prostředí.

Pojďme se nyní podívat na jednotlivé kroky této přípravy.

5.1 Instalace Pythonu

Pro získání instalačního souboru Pythonu je potřeba jej stáhnout z oficiálních webových stránek: https://www.python.org/downloads/

Instalace Pythonu je zahájena použitím staženého instalačního souboru. Během zobrazení instalačního okna byly zaškrtnuty možnosti „Use admin privileges when installing py.exe“ a „Add python.exe to PATH“. Po zvolení možnosti „Customize installation“, byly v dalším okně ponechány zaškrtnuty všechny volitelné komponenty (documentation, pip, tcl/tk and IDLE, Python test suite, py launcher, for all users).

Další krok zahrnoval výběr pokročilých možností. Některé z možností, jmenovitě „associate files with Python“, „Create shortcuts for installed applications“, „Add Python to environment variables“ a „Precompile standard library“, byly ve výchozím nastavení zaškrtnuty, a všechny byly ponechány bez změn. Z dalších možností, které nejsou ve výchozím nastavení zaškrtnuty („Install Python for all users“, „Download debugging symbols“, „Download

debug binaries“), byla zaškrtnuta pouze možnost „Install Python for all users“. Umístění instalace bylo ponecháno v defaultním nastavení, tedy `C:\Program Files\Python312`.

Instalace byla zahájena kliknutím na Install a po jejím ukončení byla využita nabízená možnost Disable PATH length limit, aby se předešlo možným problémům s omezením délky cesty v systému Windows. Tímto byla instalace Pythonu úspěšně dokončena a byl připraven základ pro další konfiguraci prostředí.

5.2 Instalace JupyterLabu

Po úspěšné instalaci Pythonu proběhla instalace JupyterLabu, s využitím správce balíčků Pythonu, pip. Ve Windows Command Prompt (cmd) byl spuštěn následující příkaz:

```
pip install jupyterlab
```

PIP automaticky stáhl a nainstaloval veškeré potřebné součásti pro fungování Jupyter Labu. Během instalace se objevila varování, upozorňující na možné problémy s nastavením cesty pro spouštěcí skripty Jupyter Labu:

```
C:\Users\Jan\AppData\Roaming\Python\Python312\Scripts is not on PATH
```

Po kontrole systémové proměnné PATH jsem zjistil, že cesty `C:\Program Files\Python312` a `C:\Program Files\Python312\Scripts` jsou již zahrnuty, ale skripty Jupyter Labu byly nainstalovány do `C:\Users\Jan\AppData\Roaming\Python\Python312`. Byly tedy nainstalovány do jiné lokality, než bylo očekáváno, a cesta k těmto skriptům nebyla automaticky přidána do systémové proměnné PATH.

Pro zajištění, že Jupyter Lab bude správně funkční a dostupný z příkazové řádky, jsem manuálně přidal chybějící cestu do systémové proměnné PATH:

Otevření ovládacích panelů -> Systém a zabezpečení -> Systém -> Upřesnit nastavení systému -> Proměnné prostředí -> Vyhledání proměnné PATH ve skupině Systémové proměnné a výběr možnosti Upravit -> Přidání nového řádku s cestou `C:\Users\Jan\AppData\Roaming\Python\Python312\Scripts` a potvrzení změn.

Po těchto krocích bylo nutné restartovat počítač, aby se změny v systémové proměnné PATH projevily a Jupyter Lab byl správně rozpoznán a dostupný přes příkazovou řádku.

5.3 Instalace nbgraderu

Pro instalaci rozšíření nbgrader byl v cmd použit příkaz:

```
pip install nbgrader
```

který zajistil stažení a následnou instalaci tohoto rozšíření. Instalace proběhla bez jakýchkoliv problémů.

Po úspěšné instalaci nbgraderu bylo nutné zajistit, že jsou všechny jeho součásti správně aktivovány. K tomu byly použity následující příkazy pro aktivaci serverových a labových rozšíření.

Aktivace serverových rozšíření pomocí příkazů:

```
jupyter server extension enable nbgrader.server_extensions.formgrader
jupyter server extension enable nbgrader.server_extensions.assignment_list
jupyter server extension enable nbgrader.server_extensions.course_list
jupyter server extension enable nbgrader.server_extensions.validate_assignment
```

Aktivace JupyterLab rozšíření skrze příkazy:

```
jupyter labextension enable nbgrader:formgrader
jupyter labextension enable nbgrader:assignment-list
jupyter labextension enable nbgrader:course-list
jupyter labextension enable nbgrader:create-assignment
jupyter labextension enable nbgrader:validate-assignment
```

Pro ověření úspěšné aktivace byly spuštěny příkazy:

```
jupyter labextension list
jupyter server extension list
```

Tyto příkazy zobrazily seznam všech aktivovaných rozšíření a potvrdily správnost instalace jednotlivých komponent nbgraderu.

Posledním krokem v procesu instalace nbgraderu je vytvoření defaultního konfiguračního souboru. Tento soubor umožňuje detailní nastavení chování nbgraderu a jeho integrace s JupyterLabem. Pro generování konfiguračního souboru se používá příkaz:

```
nbgrader generate_config
```

Tento příkaz vytvoří soubor `nbgrader_config.py` ve vašem pracovním adresáři. Konfigurační soubor obsahuje řadu nastavení, která lze upravit podle potřeb konkrétního kurzu nebo instituce. Mezi těmito nastaveními jsou například parametry pro definování struktury kurzů, specifika odevzdávání a hodnocení úkolů, a další operace spojené s managementem studentů a jejich práce.

Je doporučeno tento konfigurační soubor prozkoumat a přizpůsobit ho dle specifických potřeb výukového prostředí, aby byly všechny funkcionality nbgraderu plně využity. Úprava konfiguračního souboru umožňuje například definovat základní cesty k adresářům s materiály, nastavit způsoby bodování, nebo specifikovat, jaké informace jsou získávány od studentů při odevzdávání úkolů.

V rámci snahy o zjednodušení a zachování přehlednosti byl konfigurační soubor nbgraderu ponechán ve výchozím nastavení. Toto rozhodnutí bylo učiněno, aby se minimalizovala potřeba technických zásahů a maximalizovala uživatelská přístupnost. Důsledkem tohoto rozhodnutí je také zachování výchozí adresářové struktury, popsané v teoretické části.

Po vygenerování nebo provedení jakýchkoliv úpravách konfiguračního souboru je důležité znovu spustit JupyterLab, aby se nové nastavení projevilo.

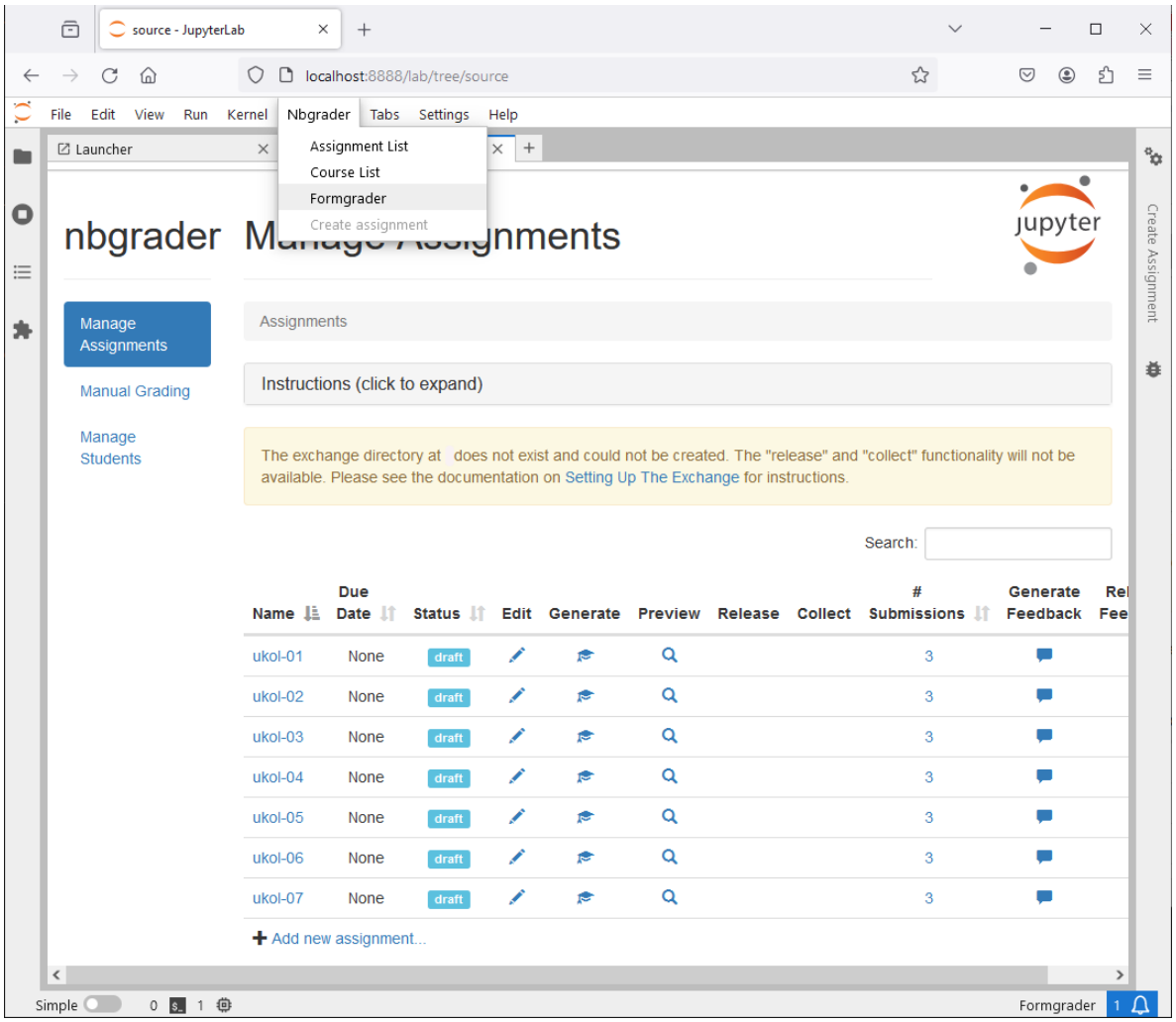
Tímto krokem je proces přípravy prostředí kompletní, a systém je připraven k použití. Spuštění JupyterLabu je možno provést příkazem:

```
jupyter lab
```

6 SPRÁVA STUDENTŮ A ÚKOLŮ

Pro efektivní správu kurzu a sledování pokroku studentů nabízí nbgrader integrované rozhraní, které umožňuje většinu funkcí pohodlně provádět přímo z uživatelského rozhraní JupyterLab.

Pro přístup k hlavnímu panelu nbgraderu je potřeba v horní navigační liště vybrat záložku „Nbgrader“ a následně kliknout na možnost „Formgrader“. Tím se otevře hlavní záložka nbgraderu, z které je možné přistupovat k jeho funkcím, včetně správy studentů, úkolů a revize odevzdaných prací.



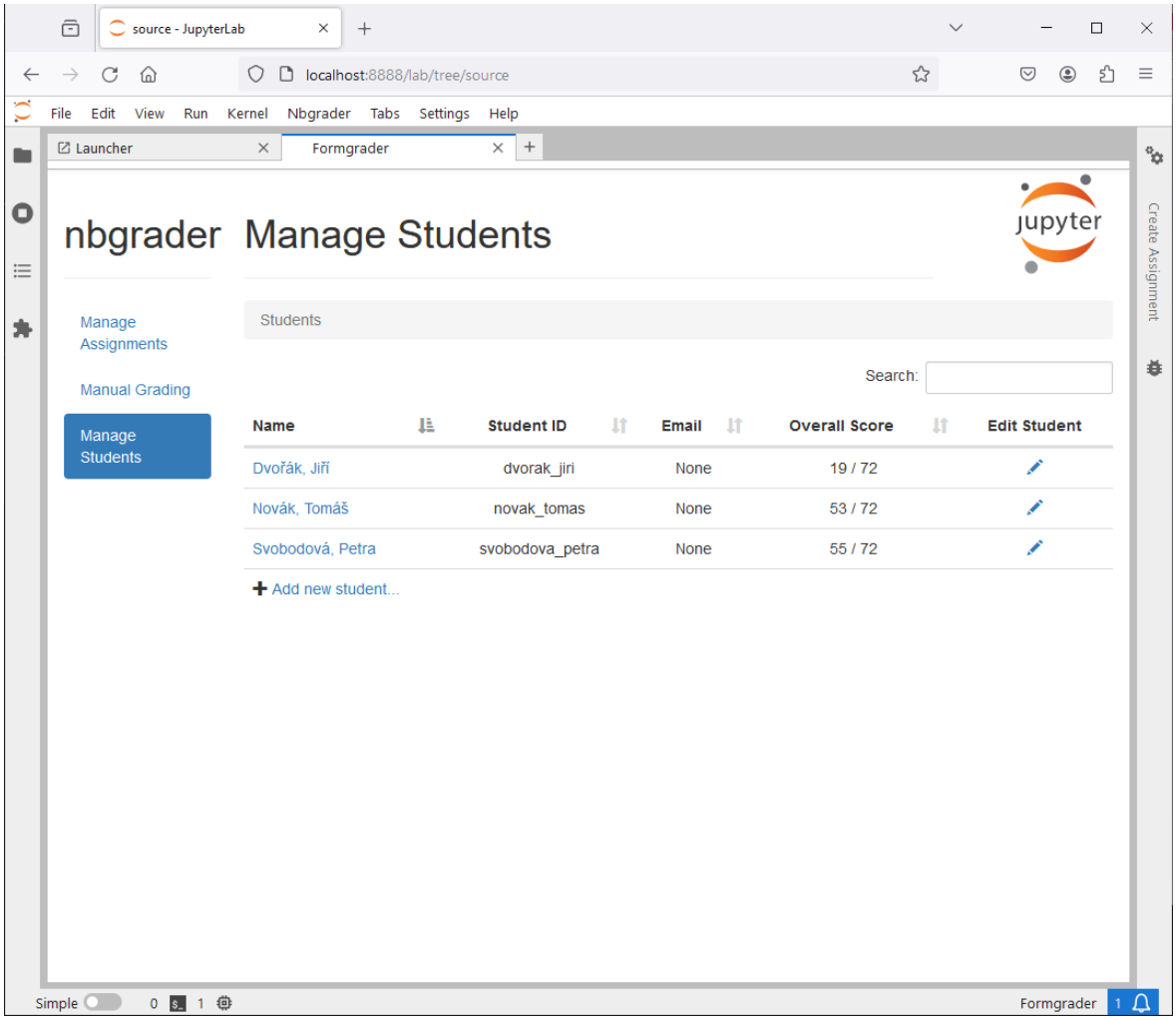
Obrázek 6. Jak se dostat do Formgraderu

6.1 Správa studentů

V levém panelu Formgraderu klikněte na možnost Manage Students, což vás přivede do sekce pro správu studentů.

Zde naleznete tlačítko Add new student, které po kliknutí otevře formulář pro zadání údajů nového studenta. Formulář obsahuje čtyři textová pole: ID (unikátní identifikátor studenta, který je povinný a po vytvoření již nelze měnit), First name (jméno studenta), Last name (příjmení studenta) a Email (emailová adresa studenta).

Po vyplnění všech potřebných údajů klikněte na tlačítko Save pro uložení údajů. Student je následně přidán do databáze nbgraderu a bude zahrnut v seznamu studentů kurzu.



Obrázek 7. Manage Students

Po přidání studenta do systému můžete jeho údaje kdykoliv upravit. V pravé části každého řádku tabulky se studenty je tlačítko Edit student, po jehož stlačení se otevře formulář pro úpravu údajů. Po provedení požadovaných změn a kliknutí na tlačítko Save se aktualizované údaje okamžitě projeví v databázi nbgraderu.

Odstranění studenta z databáze nbgraderu vyžaduje použití příkazové řádky, protože GUI tuto možnost neposkytuje. K odstranění studenta je třeba otevřít příkazovou řádku a zadat příkaz:

```
nbgrader db student remove $ID_studenta
```

V případě, že student má již ohodnocené úkoly a chcete ho přesto odstranit, je nutné k příkazu přidat parametr `--force`. Celý příkaz pak bude vypadat takto:

```
nbgrader db student remove $ID_studenta --force
```

Tento krok by měl být prováděn s opatrností, protože odstranění studenta s již ohodnocenými úkoly může ovlivnit celkovou správu kurzu a výsledky. Je doporučeno před tímto krokem provést zálohu dat, aby bylo možné data případně obnovit a předejít tak ztrátě důležitých informací.

Pro ověření, že student byl úspěšně odstraněn z databáze nbgraderu, lze v cmd využít příkaz:

```
nbgrader db student list
```

Tento příkaz zobrazí seznam všech studentů, kteří jsou aktuálně zaregistrováni v databázi.

Aby se změny projevily i v grafickém uživatelském rozhraní JupyterLabu, je nutné obnovit okno prohlížeče, ve kterém JupyterLab běží. Tím se zajistí, že data zobrazená v GUI reflektují aktuální stav databáze.

K hromadnému odstranění všech studentů z databáze nbgrader bohužel nemá přímý příkaz. K tomuto účelu se však dá použít např. následující skript využívající jiných jeho funkcí.

```
for student_id in $(nbgrader db student list | awk '{print $1}'); do
    nbgrader db student remove $student_id --force
done
```

6.2 Správa úkolů

Úkoly je možné spravovat v sekci Manage Assignments v rámci záložky Formgrader. Zde je možnost přidat úkol pomocí tlačítka Add new assignment, které otevře formulář pro vytvoření úkolu.

Formulář obsahuje pole pro název úkolu (Name), které je povinné a slouží k unikátní identifikaci úkolu v rámci kurzu. Dále je zde možnost nastavit termín odevzdání (Due date) a

časovou zónu (Timezone as UTC offset). Po vyplnění všech požadovaných údajů stačí kliknout na Save a záznam o úkolu bude přidán do databáze nbgraderu.

Po úspěšném přidání nového úkolu do systému nbgrader vznikne ve složce source nový adresář, který se jmenuje stejně jako přidaný úkol. Tento adresář slouží jako úložiště pro zdrojové notebooky spojené s daným úkolem. Notebooky v této složce obsahují jednotlivé příklady, které jsou předmětem hodnocení studentů.

Samotný proces tvorby a správy samotných notebooků bude podrobněji probrán v následující kapitole.

Editace záznamu o úkolu v nbgraderu probíhá velmi podobně jako úprava údajů studenta. Po vytvoření úkolu může nastat potřeba změnit jeho detaily, jako je například termín odevzdání. K tomuto účelu slouží tlačítko Edit assignment, které se nachází vedle každého záznamu úkolu v seznamu spravovaných úkolů.

Po kliknutí na toto tlačítko se otevře formulář, ve kterém lze upravit stávající údaje úkolu. Po provedení požadovaných změn je nutné kliknout na tlačítko Save, aby se aktualizace projevily v databázi nbgraderu.

Odstranění záznamu o úkolu z databáze nbgraderu vyžaduje obdobný postup jako odstranění studenta. Jen se používá mírně odlišný příkaz:

```
nbgrader db assignment remove $název_úkolu
```

Pokud je třeba zcela odstranit úkol, u kterého studenti již obdrželi hodnocení, je potřeba opět využít parametru `--force`

Skript pro hromadné odstranění všech úkolů z databáze může vypadat např. takto:

```
for assignment in $(nbgrader db assignment list | awk '{print $1}'); do
    nbgrader db assignment remove $assignment --force
done
```

7 TVORBA ÚKOLOVÝCH NOTEBOOKŮ

Tato kapitola se věnuje klíčovým krokům tvorby úkolových notebooků pro vzdělávací účely v prostředí JupyterLab. Jsou zde představeny metody správného strukturování notebooků a nastavení buněk, které odpovídají požadavkům automatického hodnocení a zároveň jsou srozumitelné a přístupné studentům. Dále je rozebráno efektivní využití nástrojů JupyterLabu pro správu a generování verzí notebooků určených studentům.

7.1 Vytvoření notebooku

Notebooky jsou uspořádány v adresářové struktuře, kde každý úkol má svou vlastní složku ve složce source. Např. notebook pro úkol s názvem ukol-01 bude umístěn ve složce `source/ukol-01`. K této složce se lze nejjednodušeji dostat prostřednictvím Manage Assignments v záložce Formgrader, kliknutím na název úkolu. Tím se file browser přepne do příslušné složky.

Pro vytvoření nového notebooku lze kliknout pravým tlačítkem myši v prázdném prostoru file browseru a zvolit New Notebook. Tento krok vytvoří nový notebook pojmenovaný `Untitled.ipynb`, který se automaticky otevře v editačním okně.

Po otevření notebooku vás JupyterLab vyzve k výběru kernelu. V mém případě byla zvolena možnost Python 3 (ipykernel) a zaškrtnuto Always start the preferred kernel.

Pro zachování přehlednosti je vhodné notebook pojmenovat tak, aby jeho název korespondoval s názvem úkolu (např. na `ukol-01.ipynb`).

7.2 Struktura a obsah notebooku

Notebook může obsahovat libovolné množství příkladu. Tyto příklady by však měly být navrženy tak, aby odpovídaly aktuálně probíranému tématu a aby byli bodovány na základě jejich obtížnosti.

Každý příklad v notebooku by měl být strukturován do tří základních buněk: zadání, řešení a testy. Správné nastavení těchto buněk je klíčové pro správnou funkcionalitu automatického hodnocení.

V první řadě je nutné vědět, že v JupyterLabu se dá měnit typ buňky pomocí comboboxu umístěného v levé části horní lišty editoru. Dostupné typy jsou Code, Markdown a Raw. Buňky nastavené jako Code jsou určeny pro psaní a spouštění kódu, zatímco Markdown

buňky slouží pro psaní formátovaného textu. Raw buňky zůstávají ve vygenerovaných dokumentech nezpracované, což je užitečné pro specifické formátovací účely.

Pro nastavení specifických vlastností buněk pro účely nbgraderu je třeba otevřít záložku Nbgrader Create Assignment, která se nachází v nástrojové liště napravo od editoru. V této záložce můžou lektoři nastavit typ buňky jako Read-only, Autograded answer nebo Autograded tests. Buňky nastavené jako Read-only jsou chráněny proti editaci, což je ideální pro zadání úkolů. Buňky označené jako Autograded answer slouží pro zapsání řešení příkladu studenty. Buňky určené pro testy, Autograded tests, ověřují správnost řešení a jsou rovněž chráněny před úpravami. Pokud je vybrán typ Autograded tests, může lektor navíc nastavit ještě počet bodů, které student získá za splnění úkolu.

Tato nastavení umožňují lektorům přesně definovat, jaké akce mají být s jednotlivými buňkami provedeny během hodnocení a zároveň zajistí, že studenti obdrží správně formátované a funkční notebooky.

V následujících oddílech budou detailně popsány specifika buněk pro zadání, řešení a testy.

7.2.1 Zadání

První buňkou každého příkladu by měla být buňka zadání. Ta by měla být v JupyterLab editoru formátována jako Markdown a v rámci nbgraderu označena jako Read-only. V jejím obsahu je třeba jasně stanovit zadání příkladu, aby měl student jasno v tom, co se od něho očekává. Zadání by mělo zahrnovat taky informaci o tom, kolik bodů je možné získat za splnění příkladu.

7.2.2 Řešení

Druhou částí příkladu je buňka řešení. Tato buňka má několik funkcí. Pro studenta slouží jako prostor pro vypracování příkladu. Pro lektora představuje prostor pro jeho vzorové řešení, kterým si může ověřit obtížnost vypracování příkladu a správnost napsaných testů.

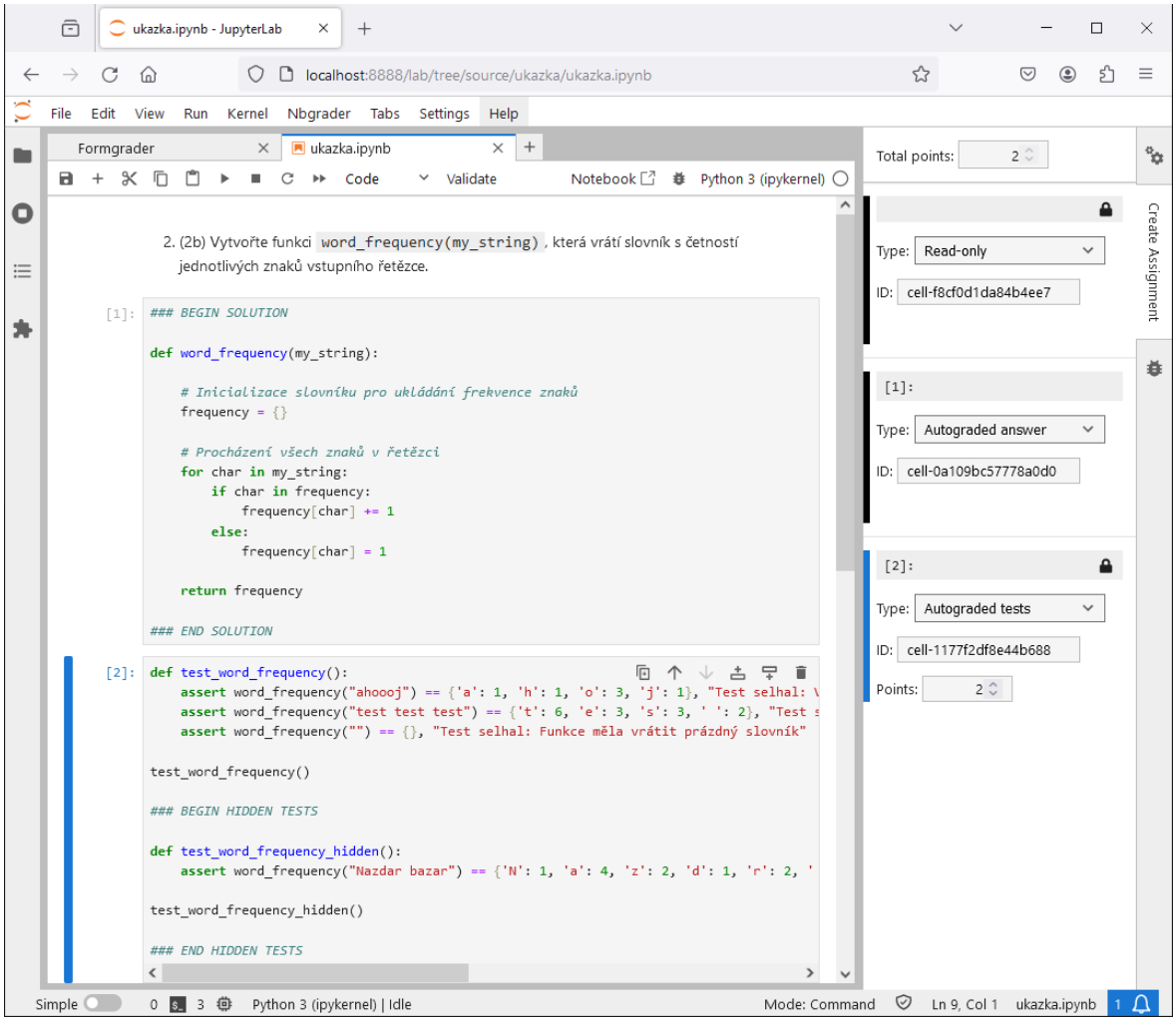
Vzorové řešení musí být ve zdrojovém notebooku ohraničeno komentáři `### BEGIN SOLUTION` a `### END SOLUTION`, to zajistí jeho odstranění při generování verze notebooku pro studenty a vložení komentáře `# YOUR CODE HERE` na jeho místo.

Tato buňka by měla být v editoru formátována jako Code a v záložce nbgraderu označena jako Autograded answer.

7.2.3 Testy

Po buňce s řešením musí následovat buňka obsahující testy, které slouží k automatizovanému ověření správnosti studentova řešení a zároveň mu poskytují okamžitou zpětnou vazbu. Testovací buňka by měla obsahovat jak otevřené, tak skryté testy, které musí být obklopeny komentáři `### BEGIN HIDDEN TESTS` a `### END HIDDEN TESTS`. Tím se zajistí že budou studentům skutečně skryty. Skryté testy slouží k dalšímu ověření správnosti řešení a jsou zásadní pro situace, kdy by se student mohl pokusit obejít systém tím, že by své řešení přizpůsobil specificky pro očekávané vstupy použité v testech. Čímž by mohl teoreticky dosáhnout správných výsledků bez porozumění problému a vypracování příkladu.

Buňka s testy by měla být v editoru označena jako Code a v záložce nbgraderu je nutné tuto buňku označit jako Autograded tests. Je také důležité nastavit v této záložce položku Points, která určuje, kolik bodů student obdrží za předpokladu, že jeho řešení úspěšně projde danými testy. Toto nastavení bodů přímo ovlivňuje, jak budou výsledky studenta hodnoceny v rámci celkového výkonu v kurzu, a motivuje tak studenty k pečlivému a promyšlenému řešení úkolů.



Obrázek 8. Ukázkový source notebook

7.3 Generování studentům určených verzí notebooků

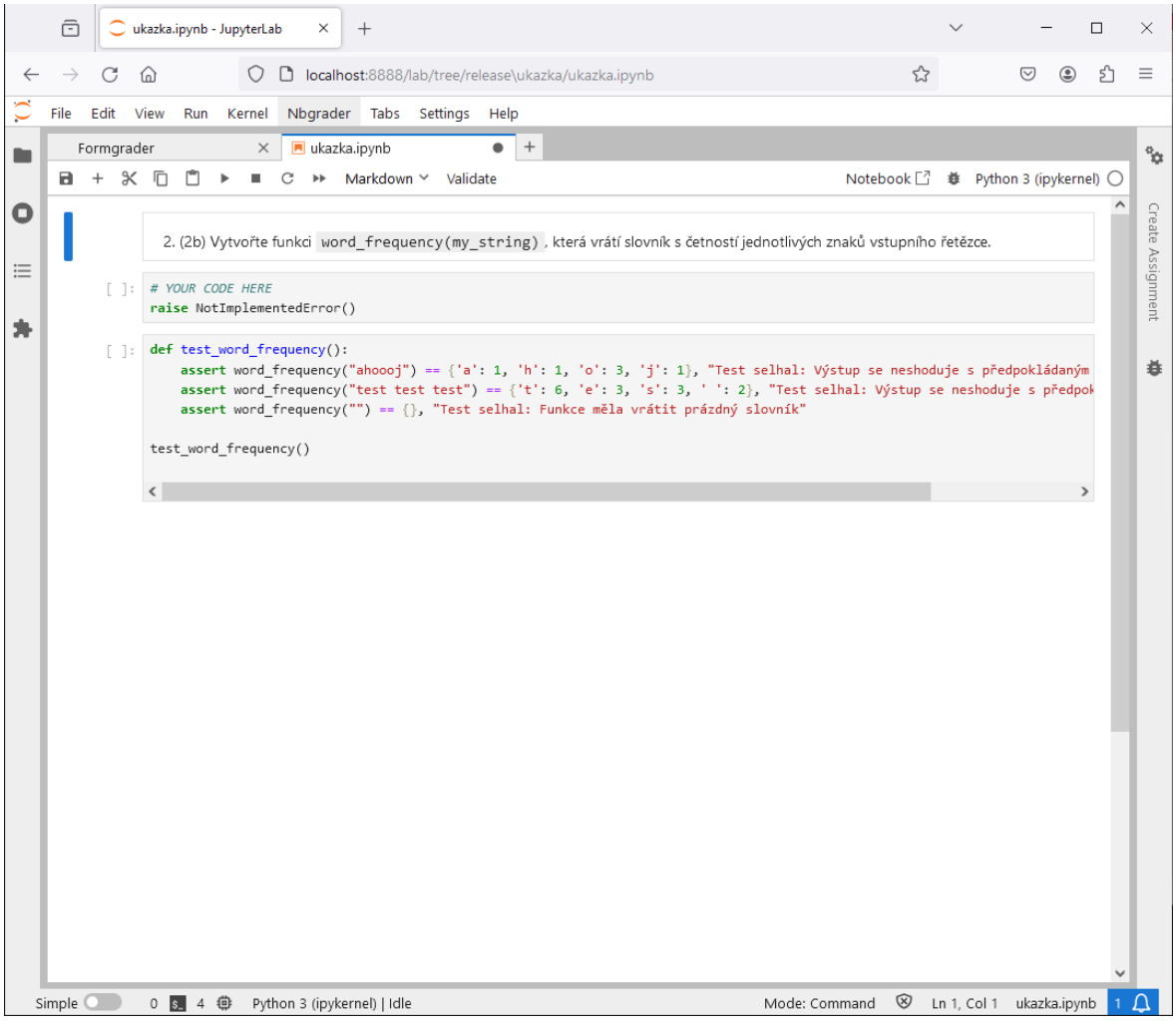
Tento proces začíná v sekci Manage Assignments, kde je možné, po vytvoření úkolu a příslušného notebooku ve složce source, vygenerovat studentům určenou verzi notebooku. K tomu slouží možnost Generate, která se nachází u vybraného úkolu v tabulce. Za předpokladu že jsou všechny buňky v notebooku správně nastaveny, a nedojde k žádnému konfliktu, uživatel po zvolení této možnosti obdrží zprávu o úspěšném vytvoření studentské verze notebooku. K tomuto nově vygenerovanému notebooku je možné přistoupit zvolením možnosti Preview u dané položky v tabulce úkolů, která přesměruje filebrowser do release adresáře daného úkolu.

Vygenerovaný release notebook se od původního source notebooku liší absencí části kódu označené jako vzorové řešení, která je v něm nahrazena řádky:

```
# YOUR CODE HERE
```

```
raise NotImplementedError()
```

A abssencí skrytých testů, aby studenti neměli přístup k plné specifikaci testů, které by mohli zneužít k obcházení systému automatického hodnocení.



Obrázek 9. Ukázkový release notebook

Tímto je soubor notebooku připraven k distribuci studentům prostřednictvím libovolné platformy, jako je Moodle, e-mail nebo jiný libovolný systém používaný ve vzdělávacím prostředí.

8 HODNOCENÍ S NBGRADEREM

Zde se zaměříme na proces hodnocení studentských úkolů v prostředí JupyterLab pomocí nástroje nbgrader. Hodnocení je klíčovým nástrojem pro měření pokroku a porozumění studentů, a proto je zásadní pochopit různé metody a nástroje, které nbgrader nabízí pro tento účel. V této kapitole se postupně seznámíme s možnostmi automatického i manuálním hodnocení v rámci nbgraderu, jakož i s možnostmi sledování výsledků na úrovni jednotlivých úkolů i celého kurzu.

8.1 Shromažďování vypracovaných úkolů

Po odevzdání úkolových notebooků studenty je nezbytné zajistit, aby byly tyto soubory řádně shromážděny pro další hodnocení. Každý vypracovaný notebook je nutné stáhnout a následně vložit do specifické struktury adresářů, která je vytvořena pro organizaci odevzdaných prací. V adresáři `submitted` má každý student svůj vlastní podadresář, označený jeho identifikačním řetězcem, obsahující adresáře pro vypracované notebooky z jednotlivých úkolů, označené názvem úkolu. Např. notebook úkolu s názvem ukol-01 odevzdaný studentem s ID dvorak_jiri by měl být umístěn do adresáře `submitted/dvorak_jiri/ukol-01/`. Tento systém organizace odevzdaných notebooků usnadňuje následné automatické zpracování a hodnocení úkolů pomocí nástroje nbgrader.

8.2 Automatizované hodnocení

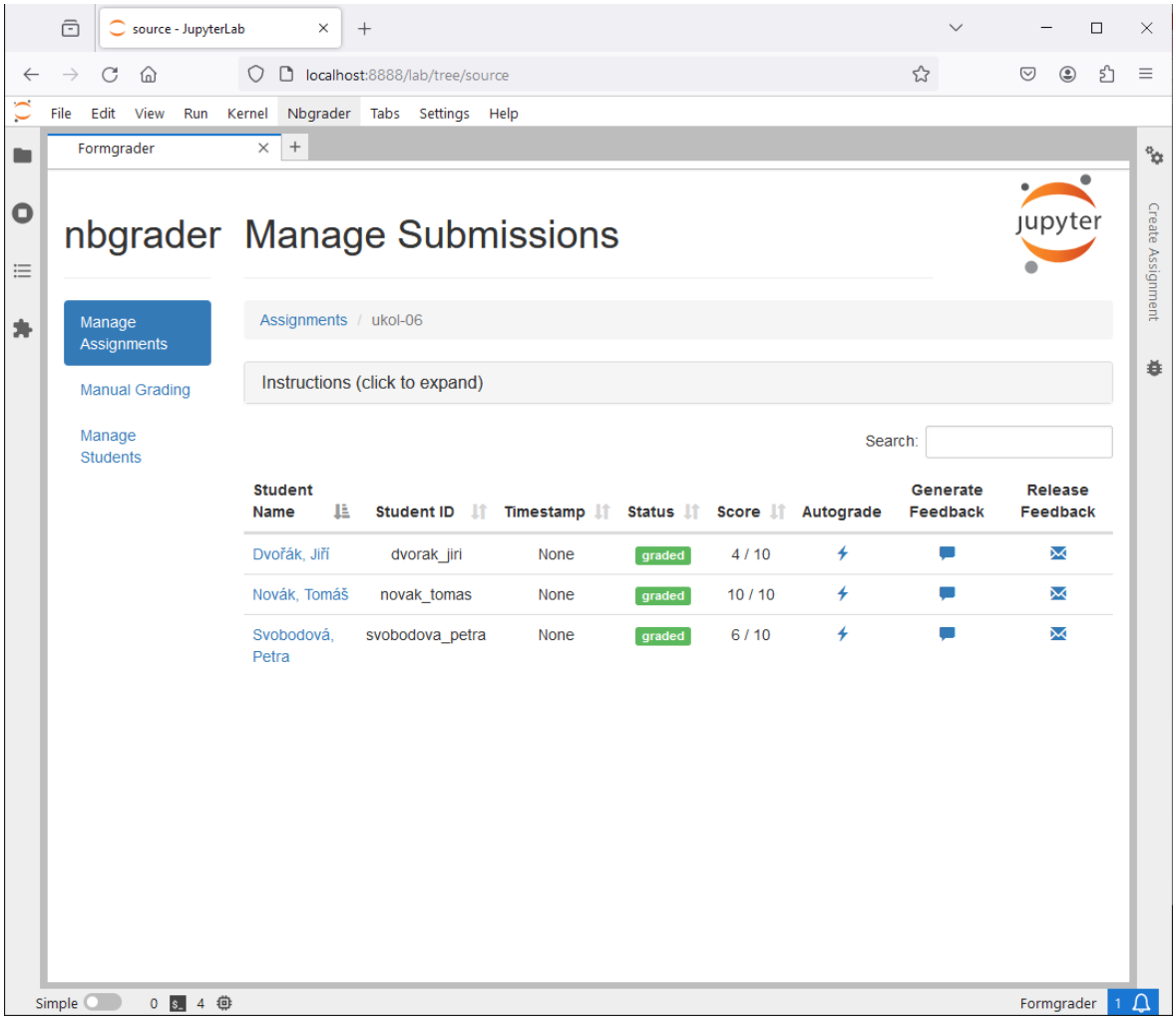
Aby bylo co nejvíce sníženo riziko vzniku problémů při automatickém hodnocení odevzdaných notebooků, je nutné, aby studenti při jejich vypracovávání dodržovali tři hlavní zásady.

První zásadou je nepřidávat nové buňky. Při automatické kontrole jsou totiž postupně vyhodnocovány všechny buňky, a pokud by student přidal novou buňku mezi řešení a testy, a v této buňce upravil definici naprogramované funkce, mohlo by to negativně ovlivnit vyhodnocení testů daného příkladu. Důvodem pro omezení přidávání dalších buněk je také udržení přehlednosti notebooku pro lektora při manuálním hodnocení.

Druhou zásadou je neměnit nastavení buněk. Jednotlivé buňky by měly být nastaveny přesně tak, jak je popsáno v kapitole 7.2 Struktura a obsah notebooku. Pokud bude nastavení některé buňky změněno, pravděpodobně dojde k chybě při automatickém hodnocení a notebook nebude možné tímto způsobem vyhodnotit.

Poslední zásadou je psát kód pouze tam, kde se nachází komentář `# YOUR CODE HERE` a nepřepisovat jiný kód než `raise NotImplementedError()`.

Proces automatizovaného hodnocení začíná v sekci Manage Assignments, kde se u každého úkolu v tabulce zobrazuje počet odevzdaných úkolů ve sloupci Submissions. Kliknutím na tento počet se záložka přepne do Manage Submissions, která obsahuje přehled všech odevzdaných notebooků pro daný úkol. V přehledné tabulce jsou zobrazeny informace o tom, který student daný úkol odevzdal, zda již byl úkol ohodnocen, a pokud ano, kolik bodů student získal.



Obrázek 10. Manage Submissions

V tabulce je také u každého odevzdaného notebooku možnost Autograde, která umožňuje automatické vyhodnocení jednotlivých úkolů. Tento nástroj je zásadní pro efektivní a rychlé hodnocení velkého množství studentů.

Pro hromadné hodnocení všech odevzdaných notebooků najednou je možné použít příkaz

```
nbgrader autograde $název_úkolu
```

Ten provede automatické hodnocení všech neohodnocených notebooků. V případě, že je třeba vyhodnotit všechny notebooky, včetně těch již ohodnocených, je možné k použitému příkazu opět přidat parametr `--force`.

Po provedení těchto příkazů je důležité v prohlížeči obnovit záložku JupyterLabu, aby se zobrazily nejnovější změny ve výsledcích hodnocení. Tento postup výrazně zjednodušuje hodnocení odevzdaných úkolů.

8.3 Manuální hodnocení

Nbgrader poskytuje také možnost manuálního hodnocení, které je aplikovatelné jak na notebooky, jež nebyly automaticky ohodnoceny, tak i na ty, které již prošly procesem automatizovaného hodnocení. Tento typ hodnocení je možné zahájit několika způsoby, přičemž nejpřímější metodou je výběr možnosti Manual Grading v záložce Formgrader. Dále v tabulce vybrat ID úkolu, který má být manuálně ohodnocen, následně zvolit ID notebooku z tohoto úkolu a poté vybrat konkrétní odevzdaný notebook.

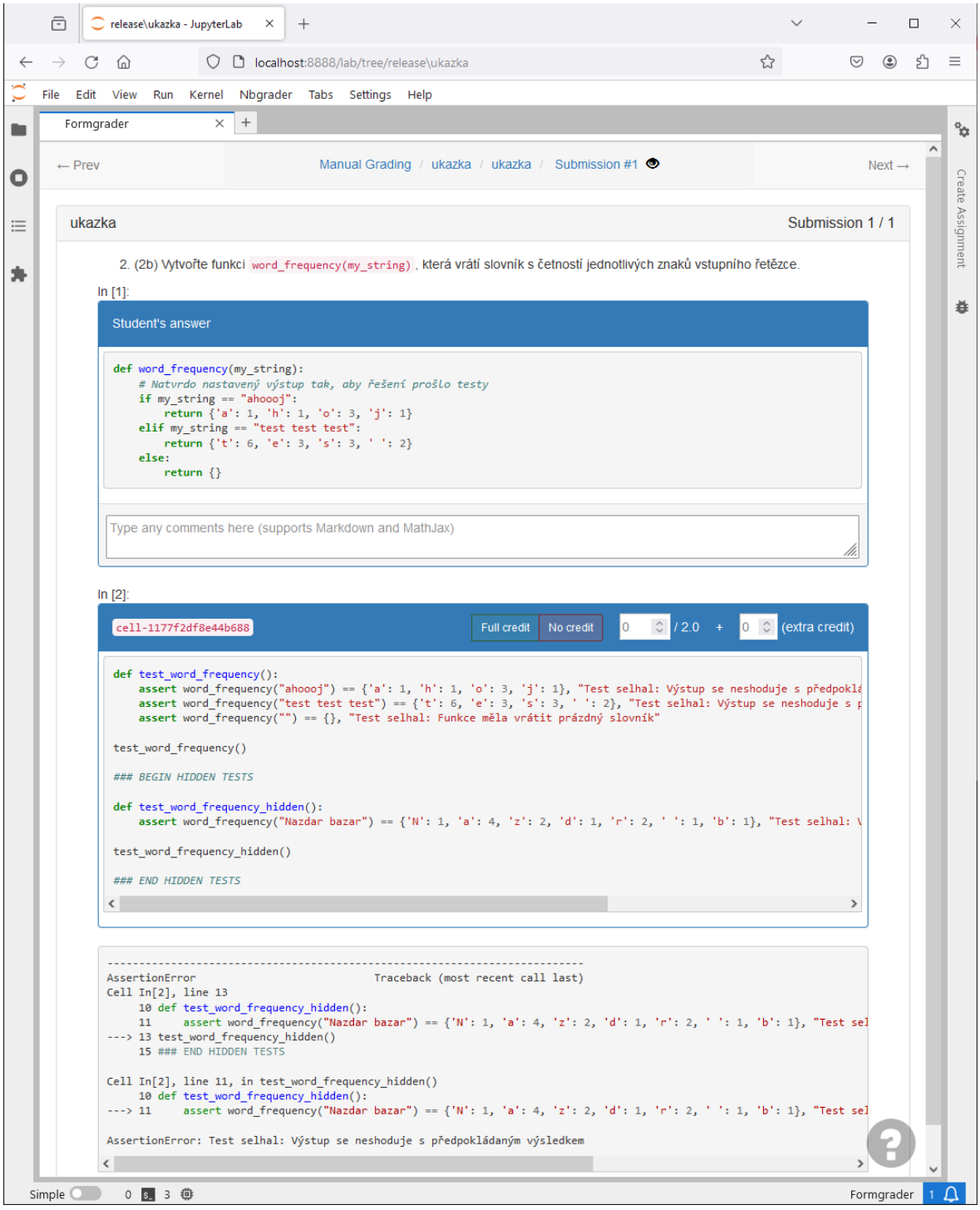
Zajímavým aspektem je, že informace o tom, kdo notebook odevzdal, není defaultně zobrazena, což podporuje objektivitu manuálního hodnocení. Pokud je však třeba zjistit identitu studenta, je k dispozici možnost Show student name.

Po výběru specifického odevzdaného notebooku se uživatelské rozhraní přepne do režimu manuálního hodnocení, kde je obsah odevzdaného notebooku zobrazen včetně skrytých testů. Lektor má možnost přidělovat body jednotlivým příkladům přímo v rámci tohoto rozhraní. Pro efektivnější navigaci mezi notebooky jsou v horní části okna umístěna tlačítka Prev a Next, která umožňují snadné a rychlé přepínání. Kliknutí na tlačítko Next, když je zobrazen poslední notebook, vrátí lektora zpět do okna s přehledem odevzdaných notebooků. To platí i pro kliknutí na tlačítko Prev, pokud je aktuálně zobrazen první notebook.

8.4 Sledování výsledků

Nbgrader umožňuje lektorům sledovat výsledky nejen na úrovni jednotlivých úkolů, ale také na úrovni jednotlivých studentů. Pro přehled o hodnocení všech odevzdaných řešení u konkrétního úkolu je možné využít sekci Manual Grading, kde se ve sloupci Score zobrazuje průměrné hodnocení daného úkolu. Po kliknutí na konkrétní úkol a výběru notebooku se

zobrazí tabulka s hodnocením všech odevzdaných notebooků tohoto úkolu, což poskytuje detailní přehled o výkonnosti studentů na úrovni jednotlivých úkolů.



Obrázek 11. Manuální kontrola automaticky ohodnoceného notebooku

Pro monitorování celkového hodnocení jednotlivých studentů slouží sekce Manage Students. V této sekci je zobrazeno souhrnné hodnocení každého studenta. Po zvolení konkrétního studenta se zobrazí tabulka se všemi jeho odevzdanými úkoly, včetně jejich hodnocení.

Tyto přehledy poskytují lektorům užitečný nástroj pro sledování pokroku studentů, což umožňuje efektivní identifikaci oblastí, kde mohou studenti potřebovat další podporu nebo vedení.

ZÁVĚR

Tato práce si kladla za cíl prozkoumat a demonstrovat možnosti využití programovacího jazyka Python v kontextu vzdělávání. Zvláštní pozornost byla věnována analýze a aplikaci nástrojů JupyterLab a nbgrader, které umožňují efektivní a interaktivní výuku a automatizované hodnocení studentů. Python se ukázal jako vhodný pro široké spektrum výukových účelů, od základních kurzů po pokročilé aplikace v datové analýze a strojovém učení, díky své přístupnosti a podpoře různých programovacích paradigmat.

JupyterLab byl využit pro tvorbu interaktivních notebooků, což umožnilo integraci výukového obsahu, kódu a testů v jednom prostředí. Nbgrader zase efektivně automatizuje hodnocení úkolů, čímž šetří čas lektorům a zajišťuje objektivitu hodnocení.

Nicméně, navrhované řešení má i své nevýhody. Instalace a správa JupyterLabu a nbgraderu vyžadují určité technické znalosti, což může být pro některé vyučující bariérou. Kromě toho, používání nbgraderu pro automatické hodnocení zahrnuje spouštění studentského kódu na lektorově počítači, což s sebou nese určitá bezpečnostní rizika.

Co se rozšíření funkcionalit navrhovaného řešení týká, mohlo by dojít k integraci s dalšími platformami, jako jsou online výukové systémy nebo Git, čímž by se dále rozšířila paleta možností využití. Zlepšení uživatelské přívětivosti instalace a správy by zase mohlo pomoci snížit technické bariéry pro vyučující. Další výzkum by se mohl také týkat zaměření na vývoj pokročilých algoritmů pro automatizované hodnocení, které by lépe rozuměly kontextu a kvalitě kódu napsaného studenty.

SEZNAM POUŽITÉ LITERATURY

[1] What is Python? Executive Summary. *Python* [online]. [cit. 2024-03-27]. Dostupné z: https://www.python.org/doc/essays/blurb/

[2] What Is Python Used For? A Beginner's Guide. *Coursera* [online]. Nov 20, 2023 [cit. 2024-03-27]. Dostupné z: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[3] PETERS, Tim. PEP 20 – The Zen of Python. *Peps Python* [online]. 19-Aug-2004 [cit. 2024-05-08]. Dostupné z: https://peps.python.org/pep-0020/

[4] HASHMI, Aamir Shahzad. The Zen of Python. *Educative* [online]. May 02, 2023 [cit. 2024-03-27]. Dostupné z: https://www.educative.io/blog/the-zen-of-python

[5] Python Changes 2014+. *Learning python* [online]. January 2022 [cit. 2024-05-08]. Dostupné z: https://learning-python.com/python-changes-2014-plus.html

[6] General Python FAQ. *Python* [online]. c2001-2024, Mar 26, 2024 [cit. 2024-03-27]. Dostupné z: https://docs.python.org/3/faq/general.html

[7] What is Python? it's Uses and Applications. *GeeksforGeeks* [online]. 26 Mar, 2024 [cit. 2024-03-27]. Dostupné z: https://www.geeksforgeeks.org/what-is-python/

[8] Lexical analysis. *Python* [online]. c2001-2024, Mar 26, 2024 [cit. 2024-03-27]. Dostupné z: https://docs.python.org/3/reference/lexical_analysis.html

[9] VAN ROSSUM, Guido, Barry WARSAW a Alyssa COGHLAN. PEP 8 – Style Guide for Python Code. *Python* [online]. 05-Jul-2001n. l., 2023-12-09 [cit. 2024-03-27]. Dostupné z: https://peps.python.org/pep-0008/

[10] Data Structures. *Python* [online]. c2001-2024, Mar 26, 2024 [cit. 2024-03-27]. Dostupné z: https://docs.python.org/3/tutorial/datastructures.html

[11] What is Python? *Amazon Web Services (AWS)* [online]. c2024 [cit. 2024-03-27]. Dostupné z: https://aws.amazon.com/what-is/python/

[12] Project Jupyter Documentation. *Jupyter* [online]. c2015 [cit. 2024-03-27]. Dostupné z: https://docs.jupyter.org/en/latest/

[13] Jupyter Notebook Documentation. *Jupyter* [online]. c2015 [cit. 2024-03-27]. Dostupné z: https://jupyter-notebook.readthedocs.io/en/latest/

[14] The Jupyter Notebook. *Jupyter* [online]. c2015 [cit. 2024-03-28]. Dostupné z: https://jupyter-notebook.readthedocs.io/en/latest/notebook.html

[15] Security in the Jupyter Server. *Jupyter Server* [online]. c2020 [cit. 2024-03-28]. Dostupné z: https://jupyter-server.readthedocs.io/en/stable/operators/security.html

[16] MAYNARD, Claire. Software testing in continuous delivery. *Atlassian* [online]. c2024 [cit. 2024-03-28]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing

[17] Co je testování softwaru? *Skillmea* [online]. 2021, 18.10.2021 [cit. 2024-03-28]. Dostupné z: https://skillmea.cz/blog/co-je-testovanie-softveru

[18] Manual Testing vs Automation Testing: Which One Should You Choose? *Testsigma* [online]. March 2, 2024 [cit. 2024-03-28]. Dostupné z: https://testsigma.com/blog/manual-testing-vs-automation-testing-which-one-should-you-choose/

[19] KINSBRUNER, Eran. Manual Testing vs. Automation Testing. *Perfecto* [online]. April 13, 2023 [cit. 2024-03-28]. Dostupné z: https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing

[20] SON, Hannah. Manual Testing vs Automated Testing: Key Differences. *TestRail* [online]. September 9th, 2023 [cit. 2024-03-28]. Dostupné z: https://www.testrail.com/blog/manual-vs-automated-testing/

[21] HLAVA, Tomáš. Fáze a úrovně provádění testů. *Testování softwaru* [online]. 2011, 21.8.2011 [cit. 2024-03-28]. Dostupné z: http://testovanisoftwaru.cz/tag/urovne-testovani/

[22] JORGENSEN, Paul. *Software testing: a craftsman´s approach*. 3rd ed. Boca Raton: Auerbach Publications, c2008. ISBN 0-8493-7475-8.

[23] SILVEIRA, Otávio Simões. A Beginner's Guide to Unit Tests in Python (2023). *Dataquest* [online]. October 6, 2022 [cit. 2024-03-28]. Dostupné z: https://www.dataquest.io/blog/unit-tests-python/

[24] Unittest — Unit testing framework. *Python* [online]. c2001-2024, Mar 28, 2024 [cit. 2024-03-28]. Dostupné z: https://docs.python.org/3/library/unittest.html

[25] Doctest — Test interactive Python examples. *Python* [online]. c2001-2024, Mar 28, 2024 [cit. 2024-03-28]. Dostupné z: https://docs.python.org/3/library/doctest.html

[26] RAMOS, Leodanis Pozo. Python's doctest: Document and Test Your Code at Once. *Real Python* [online]. Oct 31, 2022 [cit. 2024-03-28]. Dostupné z: https://realpython.com/python-doctest/

[27] Pytest: helps you write better programs. *Pytest* [online]. c2015 [cit. 2024-03-28]. Dostupné z: https://docs.pytest.org/en/8.0.x/

[28] HILLARD, Dane. Effective Python Testing With Pytest. *Real Python* [online]. Jun 22, 2022 [cit. 2024-03-28]. Dostupné z: https://realpython.com/pytest-python-testing/

[29] What is nbgrader? *Nbgrader* [online]. C2015-2017 [cit. 2024-05-08] Dostupné z: https://nbgrader.readthedocs.io/en/stable/user_guide/what_is_nbgrader.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

3D	Trojdimenzionální
CD	Continuous Delivery nebo Continuous Deployment
CDel	Continuous Delivery
CDep	Continuous Deployment
CI	Continuous Integration
cmd	Command
DevOps	Development and Operations
GIL	Global Interpreter Lock
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	Identifikace
IDLE	Integrated Development and Learning Environment
PDF	Portable Document Format
pip	Package Installer for Python
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics
Tcl	Tool Command Language
Tk	Toolkit
URL	Uniform Resource Locator

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Obsahuje 7 source a 7 release notebooků určených pro kurz Python a nástroje pro vývoj.