ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

TVORBA ELEKTRONICKÉHO VÝUKOVÉHO KURZU SE ZAMĚŘENÍM NA VÝVOJ ANDROID APLIKACÍ V JAZYCE PYTHON

DIPLOMOVÁ PRÁCE

Bc. Martin Hofman
Učitelství pro základní školy, obor Učitelství informatiky pro základní školy

Vedoucí práce: PhDr. DenisMainz, Ph.D.

Plzeň 2019

Prohlašuji, že jsem diplomovou práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

V Plzni, 29. dubna 2019

.....

vlastnoruční podpis

Poděkování

Rád bych poděkoval vedoucímu práce, kterým byl PhDr. Denis Mainz, Ph.D., a to za odborné vedené práce, mnoho věcných rad a doporučení, které mi při zpracování této práce velmi pomohly.

ZDE SE NACHÁZÍ ORIGINÁL ZADÁNÍ KVALIFIKAČNÍ PRÁCE.

OBSAH

SEZNAM ZKRATEK

APK Android Package

IDE Vývojové prostředí

ÚVOD

V současné době vlastní chytré mobilní zařízení většina lidí, proto může být pro programátora užitečné, pokud disponuje znalostmi a dovednostmi vytvářet aplikaci pro mobilní zařízení různých platforem zároveň. Cílem teoretické části práce je seznámit čtenáře se základními informacemi o programovacím jazyku Python 3, programovými nástroji na vývoj Android aplikací pomocí Pythonu a knihovny Kivy. Praktická část práce se zaměřuje na tvorbu elektronického výukového kurzu na vývoj Android aplikací v jazyce Python. Elektronický kurz je určen především pro studenty, kteří disponují znalostmi programovacího jazyka Python, ovšem pokud těmito znalostmi nedisponuje, jsou mu přímo v kurzu nabízeny možnosti k doplnění potřebných informací.

Nedílnou součást práce tvoří popis vybraných programových nástrojů jako Kivy Designer, Python IDLE a Visual Studio Code, kterým byla věnována druhá kapitola práce. Text uvedené kapitoly dále seznamuje s grafickým prostředím nástrojů a postupem jejich instalace. Pro výukový kurz byl vybrán vývojový nástroj PyCharm od velmi známého vydavatele vývojových prostředí – společnosti JetBrains. Pátá kapitola práce je zaměřena na základy vývoje Android aplikací pomocí Kivy, včetně návrhu jednodušších aplikací a vlastního doporučeného postupu vývoje aplikací. Závěr této kapitoly obsahuje podrobný postup, jak z aplikace vytvořit APK balíček pro mobilní zařízení s OS Android.

Poslední kapitola práce je zaměřena na principy objektově orientovaného programování, které jsou zde popsány teoreticky, ukázány prakticky a následně zapojeny do praktických úkolů. Některé úkoly zahrnují kompletní zadání s možným postupem řešení, kde lze výsledný zdrojový kód aplikace nalézt v příloze práce. Pro zvýšení motivace kurzem projít byl do práce zahrnut příklad tvorby komplexnějších aplikací byl do práce přidán základní článek zaměřený na SQLite databázi. Po jeho prostudování studenti kurzu přistoupí k aplikaci, kde si práci se SQLite databází vyzkouší.

1 PYTHON 3

1.1 ZÁKLADNÍ INFORMACE

Jazyk Python je velmi snadno srozumitelný jak pro čtení, tak i pro zápis. Jazyk je také velmi stručný. Díky tomu by se mohl považovat za jeden z nejsnadněji osvojitelných programovacích jazyků. Python také spadá do kategorie expresivních jazyků, což znamená, že obvykle stačí napsat menší množství kódu než v jiných programovacích jazycích (Java, C++) pro ekvivalentní aplikaci[1].

Tento jazyk vznikl již v roce 1991 a za jeho autora je považován Guido van Rossum z Nizozemska. V roce 2000 byla uvolněna verze Python 2.0 a koncem roku 2008 verze 3.0. Poslední verze je 3.7.3, která byla uvolněna 25. 3. 2019[2].

Python patří mezi multiplatformní jazyky. Program napsaný v tomto jazyce lze spustit v operačním systému Windows, OS X i systémech postavených na UNIXu. Pro spuštění na různých systémech není třeba žádná další akce jako například kompilace. Stačí veškeré soubory programu zkopírovat na cílový stroj. Jinak tomu je u systému Android a knihovny Kivy. Pokud chceme program převést na instalační balíček Androidu, tedy APK, tak máme několik možností. Jako jedna z efektivních cest se jeví sestavení pomocí nástroje buildozer. Buildozer automatizuje celý proces sestavování APK balíčku[3].

Jednu z klíčových vlastností jazyka představuje kompletní knihovna, která nám umožňuje provádět složitější programy jen na několika řádcích. Dále máme také k dispozici několik tisíc knihoven třetích stran. Nezřídka kdy jsou knihovny třetích stran výkonnější než stejná aplikace tvořená přes standardní knihovnu. Například pomocí knihovny Flask můžeme vytvořit webový server pomocí sedmi řádek kódu[4].

Python se jako programovací jazyk začal více využívat až v posledních letech. Díky informacím a jejich zpracování z Google Trends víme, že ještě v roce 2005 se o Python zajímala přibližně 3 % programátorů. Na konci roku 2018 je to už 26 %, a tím se dostala na

[1]SUMMERFIELD, Mark. *Python 3: výukový kurz*. Brno: Computer Press, 2013, s. 13-13. ISBN 978-80-251-2737-7.
[2]Python Documentation by Version. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.python.org/doc/versions/
[3]Create a package for Android. *Kivy.org* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/guide/packaging-android.html
[4]Build a Python Web Server with Flask. *Raspberry Pi* [online]. [cit. 2019-04-08]. Dostupné z: https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/2

pomyslnou první příčku v zájmu o něj. Na vrcholu je stále programovací jazyk Java. Zájem o Python vzrostl za posledních pět let o 15,8 %, na opačné straně je PHP, které za stejné období ztratilo 6,1 %[5].

Kód jazyka Python můžeme psát procedurálním i objektově orientovaným stylem.

1.2 VÝHODY JAZYKA PYTHON

Součástí této diplomové práce je také vytvoření e-kurzu pro další studenty katedry výpočetní a didaktické techniky na Pedagogické fakultě Západočeské Univerzity v Plzni. Kurz je zaměřen na vývoj Android aplikací v Pythonu.

Python má následující pozitivní vlastnosti. V první řadě velmi jednoduchá a přehledná konstrukce kódu. Python se velmi dobře čte a lze se ho velmi rychle naučit. Dokonce je považován, za nejlepší alternativu k výuce programování[6,7]. Jedna z dalších výhod je, že se tento jazyk stále více používá[8]. Posledním zásadním důvodem byla jeho velká komunita jak u nás v ČR, tak po celém světě. Více o české komunitě naleznete na webu python.cz[9].

[5]CARBONNELLE, Pierre. PYPL PopularitY of Programming Language. *PYPL* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://pypl.github.io/PYPL.html
[6]LONG, Moe. 6 Easiest Programming Languages to Learn for Beginners. *MakeUseOf* [online]. 2016, 29.12.2016 [cit. 2019-04-08]. Dostupné z: https://www.makeuseof.com/tag/easiest-programming-languages-beginners/
[7]CLEARY, Annabel. Top 5 Programming Languages for Beginners. *CoderDojo* [online]. 2015, 20.3.2015 [cit. 2019-04-08]. Dostupné z: https://coderdojo.com/2015/03/20/top-5-programming-languages-for-beginners/
[8] TIOBE Index. *TIOBE* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.tiobe.com/tiobe-index/
[9]*Python.cz* [online]. [cit. 2019-04-08]. Dostupné z: https://python.cz/

2 PROGRAMOVÉ NÁSTROJE PRO VÝVOJ ANDROID APLIKACÍ V PYTHONU

2.1 PROGRAMOVÉ NÁSTROJE

Programový nástroj si lze představit jako softwarovou aplikaci, která je vytvořena s určitým cílem. S programovými nástroji se setkáváme denně v běžném životě, a to nejen ve formě počítačových programů, ale také mobilních aplikací. Většinou existuje velká spousta programových nástrojů, které slouží ke stejnému účelu, přesto se mezi sebou mohou v mnoha ohledech lišit. Jednou z odlišností může být rychlost zpracování zdrojového kódu, dalším přívětivé uživatelské prostředí nebo také vydávání aktuálnějších a bezpečnějších verzí. Tyto aspekty mohou být velice subjektivní a záleží tedy vždy na preferencích každého člověka. Ve značné míře jsou ale preference každého člověka ovlivněny předchozími zkušenostmi. Například většina škol u nás v České republice pracuje při hodinách informatiky s programy od Microsoftu (Microsoft Word, Microsoft Excel) a lidé ty samé programy poté vyžadují i na vlastních počítačích a mobilních zařízeních, protože s nimi umí pracovat. Je třeba si ale uvědomit, že tyto programy nejsou zdarma. Ale existuje vždy mnoho alternativ, které zdarma dostupné jsou.

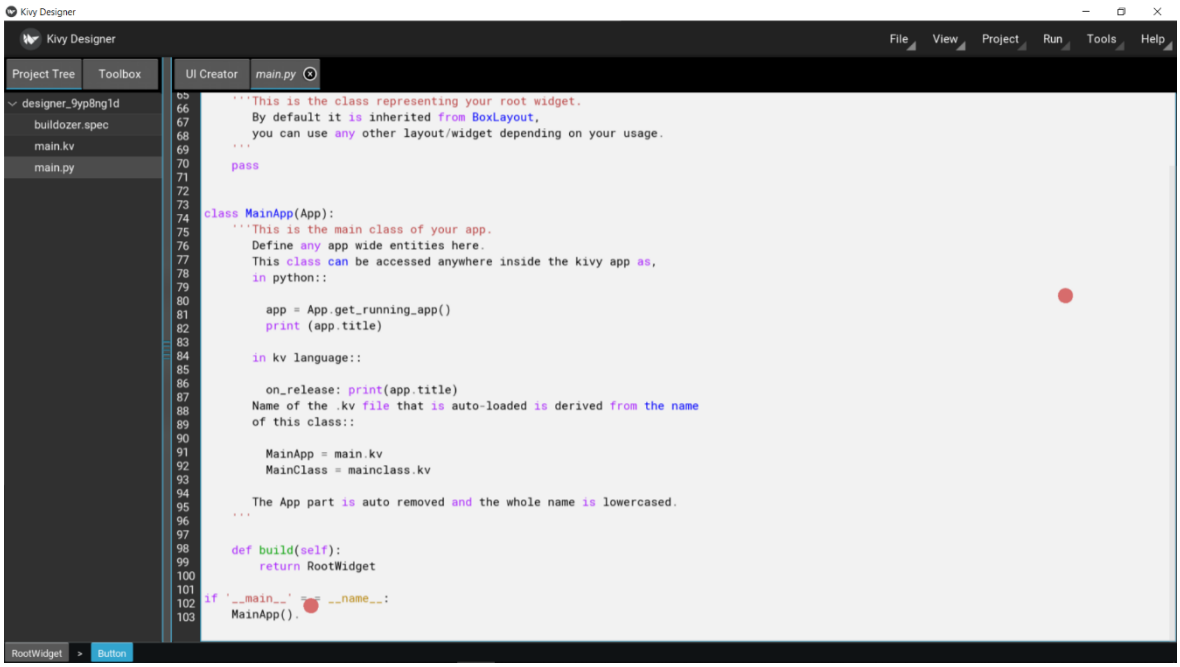
Nyní se budeme zabývat programovými nástroji pro vývoj Android aplikací v Pythonu. Bude zde uveden popis programového nástroje a jeho vlastnosti. Pro tuto práci byly vybrány programové nástroje Kivy Designer, Python IDLE a Visual Studio Code.

2.1.1 KIVY DESIGNER

Jedná se o programový nástroj primárně určený pro navrhování grafického uživatelského prostředí tvořeného prvky z knihovny Kivy. Program je kompletně vytvořený pomocí programovacího jazyka Python a je šířen pod licencí MIT (Massachusettský technologický institut)[10].

Co se týče uživatelského prostředí, to působí velmi intuitivně. Prostředí využívá jen odstínů šedé a modré barvy, které je prolínáno bílým textem. Někomu se ovšem nemusí na první pohled zamlouvat použité písmo s absencí dostatečného kontrastu s pozadím, a proto tak může působit rozostřeně. Co se týče zajištění čitelnosti zdrojového kódu, ta se s ohledem na usnadnění orientace ve zdrojovém kódu svým barevným odlišeném jeví promyšleně.

[10]Kivy Designer. *GitHub* [online]. 2018 [cit. 2019-04-08]. Dostupné z: https://github.com/kivy/kivy-designer



Obrázek 1: Ukázka prostředí Kivy Designer (zdroj: vlastní)

Z pohledu rozložení prvků se ve vrchní části nachází menu, které pro někoho možná trochu nezvykle začíná na pravé straně. Pod ním se na levé straně nachází panel vyhrazený pro stromovou strukturu projektu a skříňku s grafickými prvky, které můžeme pomocí přetažení myší vkládat do vytvářeného uživatelského prostředí. Část vždy zobrazuje pouze jednu z možností a mezi nimi se pohybujeme pomocí tlačítek v levém horním rohu. Dominantním prvkem uživatelského rozhraní je největší „středová“ část v podobě plochy zobrazující vytvořené uživatelské rozhraní, se kterým pracujeme, nebo zdrojový kód souboru. Pokud máme zobrazený zdrojový kód, tak už prostředí neobsahuje žádné další prvky. Ovšem pokud je nastaven mód zobrazení právě tvořeného grafického prostředí, objeví se ve spodní části obrazovky prostor, kde si můžeme zobrazit a upravovat zdrojový kód souboru KV, který popisuje grafickou stránku aplikace. Další možností daného prostoru je zobrazení konzole Kivy, konzole pro chyby anebo příkazová řádka Pythonu. Následujícím prvkem uživatelského prostředí je panel, který se nachází na pravé straně obrazovky. Vrchní část panelu nám umožňuje nastavit velikost obrazovky v pixelech a přibližování nebo oddalování pohledu. Prostřední část panelu obsahuje stromovou strukturu grafických prvků a pod ní se nachází prostor pro nastavování vlastností a událostí grafických prvků. Posledním prvkem je spodní lišta, která zobrazuje stromovou strukturu pro prvek, s nímž je aktuálně pracováno.

S ohledem na výkon pracuje Kivy Designer svižně. To lze v dnešní době na běžných strojích očekávat. Ovšem je nutné zároveň dodat, že tento program značným způsobem vytěžuje procesor. Na stroji s procesorem Intel i5 – 3210M se dvěma jádry a čtyřmi logickými procesory, využívá téměř 30 % jeho výkonu a tato hodnota není závislá na tom, zda v Kivy Designeru pracujeme či nikoliv. Ostatní nároky jako např. paměť jsou dle očekávání minimální.

Na webových stránkách[11], kde je Kivy Designer k dispozici, je uvedeno, že se jedná o nestabilní alfa verzi, která zatím není vhodná pro obecné použití. Také je zde uvedeno, že aktuálně neexistuje další plán vývoje tohoto programového nástroje.

Instalace Kivy Designeru je poměrně jednoduchá, ovšem je nutné mít správně nastavené systémové a uživatelské proměnné. Konkrétně u proměnné PATH a PYTHONPATH:

- C:\users\<Username>\Local Settings\Application Data\Programs\Python\Python 36
- C:\users\<Username>\Local Settings\Application Data\Programs\Python\Python 36\Scripts
- C:\users\<Username>\Local Settings\Application Data\Programs\Python\Python 36\Lib
- C:\users\<Username>\Local Settings\Application Data\Programs\Python\Python 36\DLLs
- C:\users\<Username>\Local Settings\Application Data\Programs\Python\Python 36\lib\site-packages

Nyní je systém připraven na instalaci Kivy a Kivy designeru. Po jejich instalaci je nutné ještě získat a instalovat aplikaci GIT (distribuovaný systém správy verzí). Posledním krokem je nastavení systémové proměnné GIT_PYTHON_GIT_EXECUTABLE na:

- C:\Program Files (x86)\Git\cmd\git.exe

V tuto chvíli by se měl Kivy Designer bez problémů spustit pomocí příkazu: python -m designer.

Největší předností Kivy designeru je vytváření grafického uživatelského prostředí metodou Drag and Drop (způsob, kdy je nad prvkem stisknuto tlačítko myši, za stálého držení tlačítka je prvek přetažen na výsledné místo a následně je tlačítko uvolněno). Tento

[11]Kivy Designer. *GitHub* [online]. 2018 [cit. 2019-04-08]. Dostupné z: https://github.com/kivy/kivy-designer

způsob je určitě efektivnější než vytváření samotného kódu pomocí jazyka Kivy v souboru typu KV.

2.1.2 PYTHON IDLE

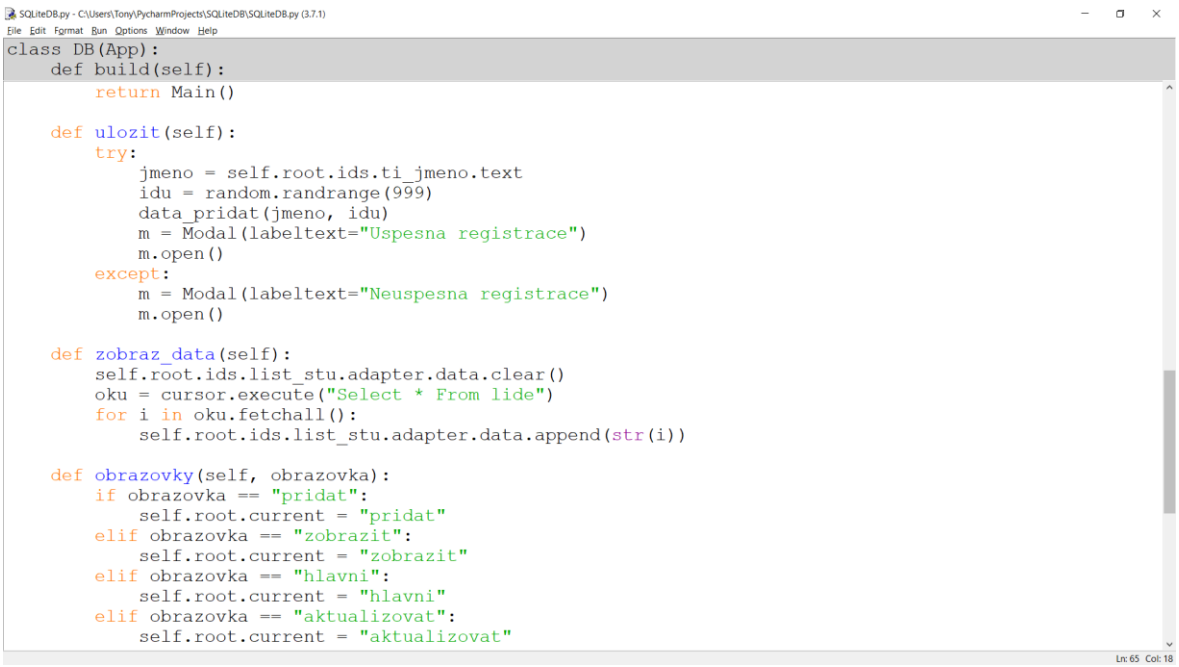
Tento programový nástroj je primárně určen pro vývoj a studium v Pythonu a sám je pomocí Pythonu naprogramován. Aktuální verze Pythonu a Pythonu IDLE je 3.7.3, která byla uvolněna 25. března 2019[12]. Samotné IDLE má následující funkce:

- kódování v čistém Pythonu
- multiplatformní kód – většinou funguje na Windows, Unixu i MacOS
- python shell – interaktivní interpretace s barevným odlišením vstupů, výstupů a chybových hlášek
- textový editor – s více okny, barevným rozlišením kódu, inteligentním odsazením, tipy pro volání a automatickým dokončováním
- hledání v libovolném okně
- ladicí mód – s trvalými zarážkami, krokováním a zobrazováním obsahu proměnných[13]

Pokud se zaměříme na uživatelské prostředí, tak se opravdu jedná o vylepšený textový editor. V horní části prostředí je umístěna nabídka funkcí s položkami **File** (vytváření a otevírání souborů, různé možnosti ukládání i zavírání souborů a také úplné ukončení editoru), **Edit** (vrátit/posunout krok, kopírování, vkládání a výběr textu, vyhledávací i nahrazující funkce a volání tipů), **Format** (zejména pro práci s odrážkami), **Run** (spuštění python shellu nebo aplikace), **Options** (pro nastavení IDLE), **Window** (roztažení okna na celou výšku obrazovky a správu otevřených oken IDLE) a **Help** (informace o Pythonu i IDLE a přístup k dokumentaci Pythonu).

[12]Python Documentation by Version. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.python.org/doc/versions/

[13] IDLE. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://docs.python.org/3/library/idle.html



Obrázek 2: Ukázka prostředí Python IDLE (zdroj: vlastní)

Samotná část pro zdrojový kód je velmi jednoduchá. Kód se dobře čte a je barevně rozlišen podle kategorií. Názvy tříd a funkcí jsou modré barvy, klíčová slova jazyka oranžová a řetězce jsou barvy zelené. Částečně neefektivní se může stát práce na větším projektu, protože každý soubor je pak otevřen ve svém vlastním okně editoru.

S ohledem na potřebný výkon, který aplikace vyžaduje pro svůj běh, spotřebuje editor maximálně 5 % výkonu CPU při třech souběžně spuštěných oknech editoru. Editor funguje rychle a jeho instalace se provede při samotné instalaci Pythonu a není již třeba žádná další konfigurace stroje.

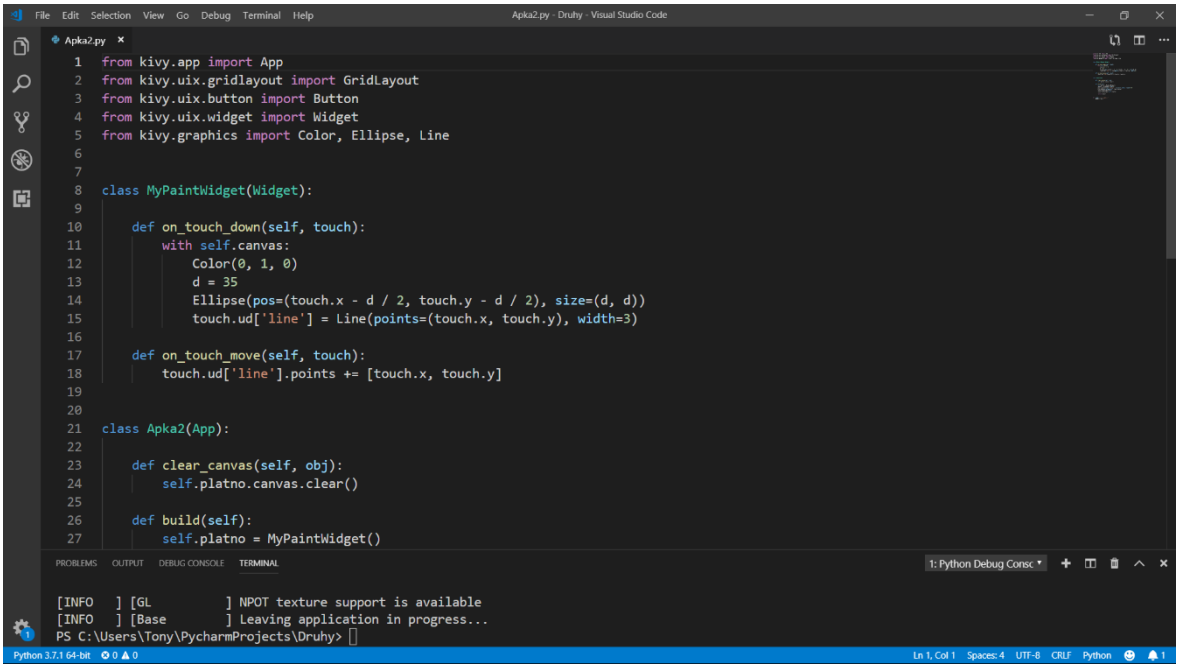
2.1.3 VISUAL STUDIO CODE

Jedná se o lehký, ale výkonný editor zdrojového kódu, který je dostupný ve verzi pro Windows, Linux i MacOS. V základní verzi má zabudovanou podporu JavaScriptu, TypeScriptu a Node.js, ale existuje bohatá sada rozšíření pro další skriptovací a programovací jazyky. Zásadní je podpora pomocí rozšíření pro jazyk Python, Kivy a KV. Aktuální je verze 1.33, která byla uvolněna v březnu 2019[14].

Na první pohled vypadá prostředí velmi propracovaně. Důležité části jsou jasně odděleny a kontrast mezi prvky, textem a pozadím je výborně zpracován. Prostředí budí dojem

[14]March 2019 (version 1.33). *Visual Studio Code* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://code.visualstudio.com/updates/v1_33

opravdu profesionálně zpracovaného programového nástroje. Výborně je zpracována identifikace zdrojového kódu, který je velmi dobře barevně odlišen podle kategorií.



Obrázek 3: Ukázka prostředí Visual Studio Code (zdroj: vlastní)

Z hlubšího pohledu se nyní podíváme na rozložení prostředí. V levé horní části se standardně nachází nabídka, která nám umožňuje pracovat s projekty, soubory a jednotlivými obsahy souborů. Nabídka dále zahrnuje možnosti pro zobrazení součástí prostředí, spuštění aplikace včetně ladicího módu. Na levé straně se nachází panel, který má svoji vlastní nabídku položek zobrazujících např. data či jiné možnosti. Zásadní položkou nabídky je rozšíření, které umožňuje připravit Visual Studio Code pro vývoj aplikací v Pythonu. Nejvíce používanou položkou při programování bude prohlížeč, který nám nejen ukazuje obsah projektu, ale také seznam otevřených souborů. Samozřejmostí je také ladicí mód, bez kterého se programátor takřka neobejde. Hlavní část okna je vyhrazena pro zdrojový kód. Pod hlavní částí se ve vlastní sekci zobrazují chyby a upozornění. V této časti lze ještě přepnout na výstup, ladicí konzoli a terminál. Nutno podotknout, že postranní panel a prostor pod hlavním oknem lze jednoduše skrýt. Posledním prvkem je spodní lišta, která nás nalevo informuje o tom, kterou verzi Pythonu k interpretaci projektu používáme, zda zdrojový kód obsahuje nějaké chyby nebo jsou zde zobrazována další upozornění. V její pravé části je zobrazeno, na které řádce a znaku se nacházíme, počet mezer, který identifikuje úroveň kódu, použité kódování a další již méně podstatné informace.

Velmi dobře funguje nástroj zvaný IntelliSense, který nejen zvýrazňuje klíčové prvky kódu, ale také automaticky doplňuje kód, nabízí následující možnosti na základě typů proměnných, definic funkcí a importovaných modulů.

Prostředí funguje rychle a jen minimálním způsobem využívá prostředků procesoru. Prostředky v podstatě využívá jen, když se v editoru přímo pracuje, a vystačí si s necelými 5 % výkonu procesoru. Instalace Visual Studio Code je velmi snadná, stačí si pouze stáhnout instalační balíček z oficiálních stránek poskytovatele a pomocí průvodce software nainstalovat.

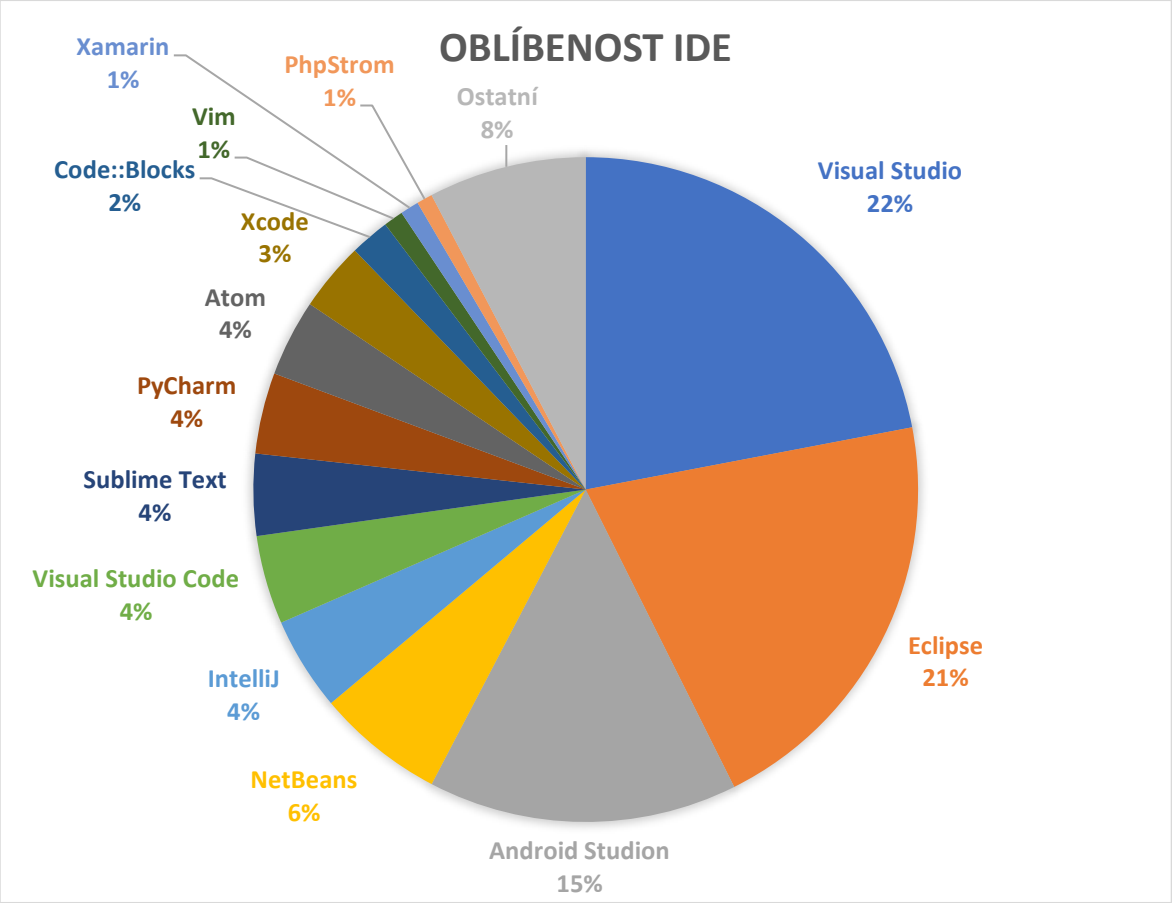
3 PyCharm

Vývojové prostředí je velmi sofistikované a uživatelsky přívětivé stejně tak, jako ostatní produkty od společnosti JetBrains. Tato společnost se na trhu objevila roku 2001, kdy otevřela svoji první pobočku u nás v ČR, konkrétně v Praze. Prvním produktem bylo prostředí IntelliJ, které je určeno pro vývoj Java aplikací. V současné době má společnost 6 poboček, kde pracuje přibližně tisíc zaměstnanců a poskytuje více než 20 produktů[15].

3.1 Zdůvodnění výběru vývojového prostředí PyCharm

Hlavním důvodem výběru prostředí PyCharm bylo, že toto vývojové prostředí velmi inteligentně reaguje na potřeby programátora při jeho práci s prostředky, mezi které patří kontrola v reálném čase, kontextová nápověda a rychlost zpracování. Prostředí je nabízeno i ve verzi pro bezplatné využití. Jedním z dalších důvodů bylo, že společnost JetBrains nabízí vývojová prostředí pro mnoho programovacích jazyků, ale v podstatě je staví na jednom konceptu, a proto je velmi jednoduché změnit programovací jazyk, ale zůstat v podstatě v jednom prostředí. Tento princip je v souladu s jazykovou flexibilitou požadovanou po programátorech v dnešní komerční praxi. Jedním z dalších důvodů výběru nástroje bylo, že toto vývojové prostředí používá stále více programátorů. To dokládají informace na webu pypl.github.io/IDE.html, kde je toto prostředí aktuálně na 7. pozici v žebříčku oblíbenosti vývojových prostředí[16].

[15]About Company. *JetBrains* [online]. [cit. 2019-04-08]. Dostupné z: https://www.jetbrains.com/company/
[16]CARBONNELLE, Pierre. Top IDE index. *PYPL* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://pypl.github.io/IDE.html



Obrázek 4: Oblíbenost vývojových prostředí (zdroj: vlastní)

Pycharm používají i ve společnostech jako HP, Symantec a Twitter. Společnost JetBrains dlouhodobě dokazuje, že její vývojová prostředí patří k těm nejlepším. Za posledních 15 let společnost získala více než 50 různých ocenění[17].

3.2 Získání vývojového prostředí a jeho instalace

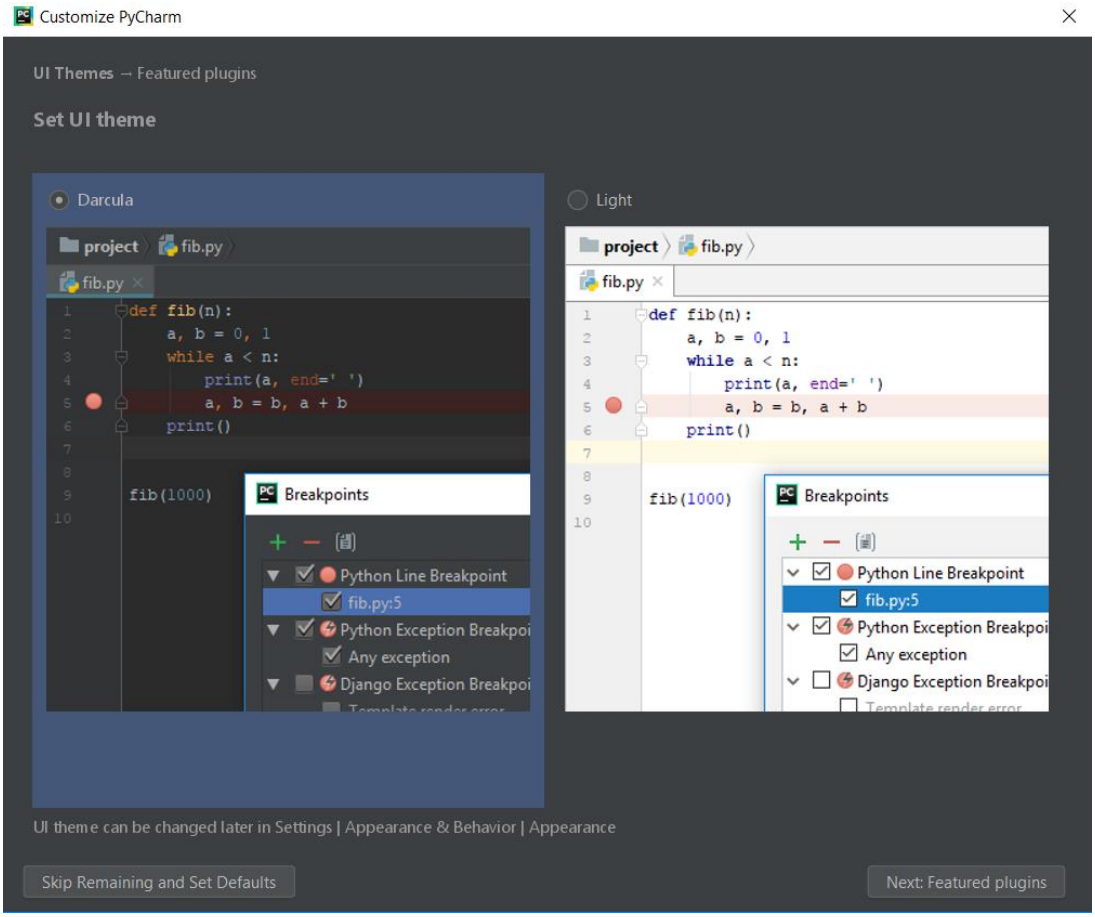
Pycharm je k dispozici ve dvou verzích, a to Community a Professional. Verzi Community můžeme používat zdarma. Ta nám poskytuje Python editor, grafický ladicí program a zkušební spuštění, refactoring, kontrolu kódu a kontrolu verzí projektů. Verze Professional je k dispozici zdarma v rámci 30-ti denního trialu. Ta navíc oproti verzi Community obsahuje rozšíření pro navrhování webů, webové frameworky pro Python a hlavně podporu databází. Vyšší verzi mohou získat studenti i učitelé zdarma na jeden rok pro vzdělávací účely. Pro ověření příslušnosti ke vzdělávací instituci je upřednostňován školní/univerzitní email nebo mezinárodní studentský průkaz ISIC. Potvrzujícím emailem obdrží zájemce odkaz na registraci účtu, ke kterému lze následně

[17] About Customers & Awards. *JetBrains* [online]. [cit. 2019-04-08]. Dostupné z: https://www.jetbrains.com/company/customers/

dostat licenční číslo pro Pycharm Professional, ale i mnoho dalších produktů společnosti JetBrains.

Prostředí je k dispozici jak pro Windows, tak pro Linux i macOS. Systémové požadavky jsou minimální a splňuje je každý běžně prodávaný počítač i notebook. Pycharm by měl spolehlivě fungovat na operačním systému Windows XP a každé novější verzi. Minimální operační paměť je stanovena na 2 GB, ale doporučují se 4 GB. Verze Community má při stahování velikost 208 MB a při instalaci vyžaduje téměř 500 MB diskového prostoru. Instalace Probíhá pomocí standardního průvodce s možností volby cesty k adresáři instalované aplikace. Na další obrazovce se průvodce dotazuje, zda má být vytvořen zástupce na ploše a jestli má být provedena asociace k souborům s příponou .py. Následně proběhne samotná instalace, která na běžném stroji zabere jen několik minut.

Pokud proběhne instalace v pořádku, tak je vývojové prostředí připraveno k prvnímu spuštění. Při prvním spuštění dojde k zobrazení průvodce nastavení. První část průvodce je zaměřena na uživatelské prostředí. K výběru vzhledu prostředí se nabízí tmavá (Darcula) nebo světlá (Light) varianta. Za výchozí se dnes považuje tmavé uživatelské rozhraní, i když tomu tak dříve nebylo. V dalším kroku průvodce je možnost instalace nejčastěji instalovaných pluginů (zásuvných modulů). V posledním kroku se nám již spustí samotné vývojové prostředí. Nastavení zvolená v úvodu lze později změnit.



Obrázek 5: Výběr uživatelského rozhraní (zdroj: vlastní)

3.3 Vlastnosti GUI

Grafické rozhraní vývojového prostředí je velmi přehledné a intuitivní. Zejména propracované je členění prostředí. Programátor si může prostředí téměř libovolně upravovat. Ve výchozím nastavení je na levé straně struktura projektu, uprostřed okno pro kód otevřeného souboru a ve spodní části obrazovky je vyhrazena část pro vypisování událostí. Všem těmto částem lze změnit velikost, či je úplně skrýt. Jediné pevné části tvoří spodní lišta a pás karet hlavní nabídky, pod nímž je zobrazena cesta k otevřenému souboru v projektu. Největší vliv na zobrazení částí projektu má tlačítko umístěné vlevo na spodní liště.

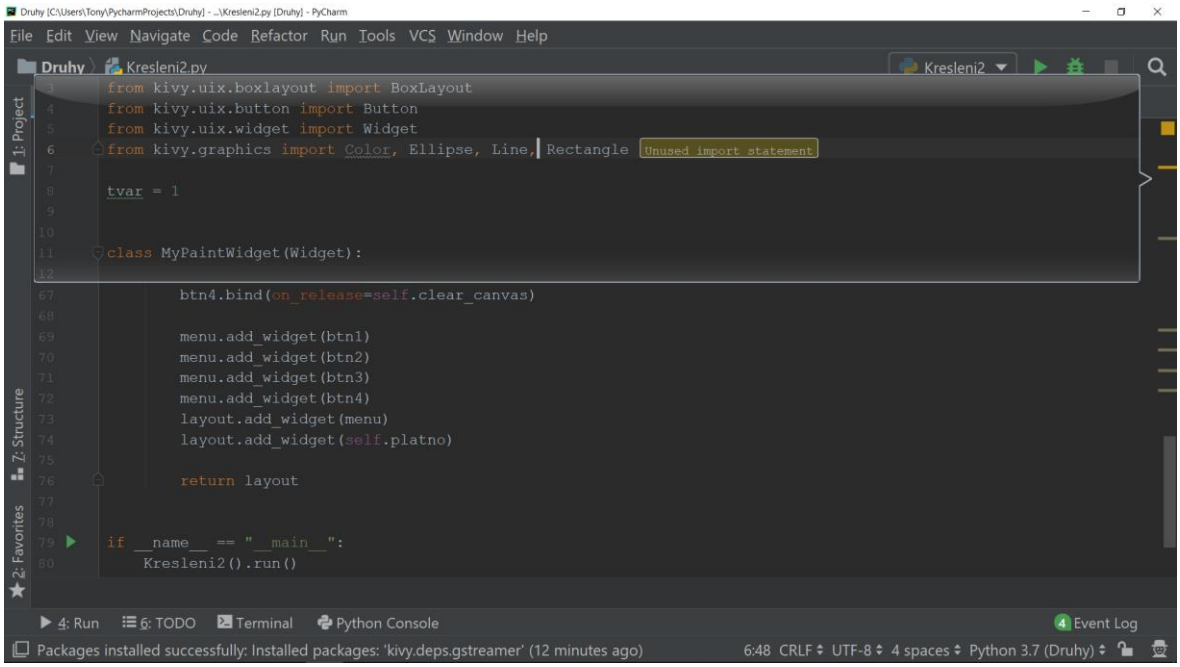
Obrázek 6: Tlačítko pro úpravu GUI (zdroj: vlastní)

Po jeho stisknutí se na boční a spodní straně zobrazí požadované podokno. Většinou se jedná o skupinu tlačítek, která sdílejí jedno podokno. V podokně se zobrazí vybraná položka, případně i více vybraných. Díky tomu uživatel nemusí dlouze hledat v hlavní

nabídce, kde je možnost si danou část zobrazit, a tím šetří čas při vývoji aplikací. Samotná část pro strukturu projektu je přehledně uspořádaná a části projektu jsou od sebe snadno rozeznatelné. Podstatné části aplikace mají různé ikony.

Velmi dobře je také propracované podokno pro psaní zdrojového kódu. Barevně jsou zde odděleny skupiny klíčových slov. Pokud má uživatel aktivní tmavé uživatelské rozhraní (Darculatheme), tak jsou klíčová slova jazyka zobrazována světle oranžovou barvou (#CC7832), názvy metod okrovou barvou (#FFC66D), textové řetězce zelenou barvou (#6A8759), instance světle fialovou barvou (#9876AA) a ostatní kód šedomodrou barvou (#A9B7C6).

Prostředí také obsahuje funkci pro zobrazení kódu dalších částí otevřeného souboru. Funkce je součástí posuvné lišty. Programátor si na běžném monitoru pohodlně zobrazí kolem padesáti řádek kódu. Ovšem pokud v prostředí Pycharm přesune kurzor myši na posuvnou lištu, zobrazí se podokno s deseti řádky kódu z místa umístění myši na liště. Snadnými pohyby si uživatel může prohlédnout část kódu, kterou má napsanou na jiném místě v souboru. Nejenže se zobrazí kód z jiné části, ale jsou tam zobrazena také doporučení pro provedení úprav kódu[18].



Obrázek 7: Ukázka prostředí a funkce pro zobrazení kódu v nezobrazené části (zdroj: vlastní)

[18]HOFMAN, Martin. *Vývoj Android aplikací v prostředí IntelliJ*. Plzeň, 2017. Bakalářská práce. Západočeská univerzita v Plzni.

4 KIVY

Kivy je volně dostupná knihovna pro programovací jazyk Python. Kivy nám umožňuje navrhovat aplikace pro Windows, Linux, Android, iOS a macOS. Dokonce máme i knihovnu KivyPie, která je určena pro Raspberry Pi. Kivy je v podstatě moderní nástroj pro grafické uživatelské rozhraní, pod licencí Massachusettského technologického institutu (MIT). Kivy podporuje více dotykové vstupní zařízení a také hardwarovou grafickou akceleraci[19].

4.1 ZÍSKÁNÍ KIVY A JEHO INSTALACE PRO PYCHARM

Instalace Kivy je uvedena přímo na stránkách kivy.org, ale tento postup není vhodný pro uživatele PyCharmu. Pro používání Kivy přímo ve vybraném vývojovém prostředí musíme postupovat následovně (za předpokladu, že již máme nainstalovaný samotný Python):

1. Spustit PyCharm a přes menu otevřít File -> Settings (případně Ctrl + Alt + s) nastavení. V okně pro nastavení je v levé části záložka Project a v ní záložka Project Interpreter, kterou otevřeme.
2. Ve stejném okně stiskneme v pravé horní části tlačítko „+“. Reakcí by mělo být otevření okna s dostupnými balíčky.
3. V levém horním rohu nalezneme okénko pro vyhledávání pomocí filtru, kterého zadáme klíčové slovo „kivy“, díky čemuž se vyfiltrují dostupné balíčky.

   Postupně nalezneme a myší vybereme další balíčky. Poté stiskneme v levém dolním rohu tlačítko „Install Package“. Potřebné balíčky mají označení: Kivy, kivy.deps.sdl2, kivy, deps.glew, kivy.deps.gstreamer a kivy.deps.angle.

4. Pokud instalace proběhla v pořádku, můžeme okno zavřít a práce je hotova. Častým problémem selhání instalce je neaktuální verze PyCharmu. Po provedené aktualizaci prostředí by mělo být možné postup zopakovat již úspěšně.

Výsledně je možné vidět, že v PyCharmu přibylo více balíčků, než které byly před instalací vybrány. Důvodem je to, že instalované balíčky jsou závislé na dalších, které se nainstalují automaticky.

4.2 PODPORA PŘI VÝVOJI APLIKACÍ

Knihovna Kivy je v dnešní době velmi známá a také používaná. Pokud se tedy rozhodneme tuto knihovnu používat, máme k dispozici nemalé množství zdrojů. O internetové zdroje se stará zejména komunita, která se zajímá o samotný vývoj Python aplikací s knihovnou

[19]Kivy: Cross-platform Python Framework. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/#home

Kivy. Pokud vezmeme v potaz tutoriály, tak těch textových i video nalezneme mnoho, ale velká většina jich je v anglickém jazyce. Jediný český tutoriál poskytuje portál ITnetwork.cz. Tento tutoriál se skládá z devíti lekcí, kde poslední tři lekce již nejsou zcela zdarma a je třeba zaplatit poplatek pro jejich kompletní zobrazení.

Zdrojů se k dispozici nabízí dostatečné množství, ale pro zajištění větší pohodlnosti při práci s nimi je dobré být odpovídajícím způsobem jazykově vybavený. Další problém většiny nalezených kurzů spočívá v absenci propojení práce s knihovnou Kivy a objektově orientovaného přístupu při programování.

Kurzy a tutoriály ovšem nejsou jediným zdrojem informací. Dalším zdrojem informací je samotná dokumentace ke knihovně Kivy. Dokumentace je velmi rozsáhlá a u každého prvku zjistíme, jak s daným prvkem pracovat, jak ho implementovat a upravit jeho vlastnosti. Pro někoho může být tato dokumentace primárním zdrojem, ale také nemusí.

Pokud nahlédneme do světa literatury, tak existuje hned několik knih, které se zabývají knihovnou Kivy:

- Kivy: Interactive Applications in Python, Roberto Ulloa, 2015
- Kivy Cookbook, Hugo Solis, 2015
- Kivy Blueprints, Mark Vasilkov, 2015
- Creating Apps in Kivy: Mobile with Python, Dusty Phillips, 2014

Rozhodně nelze opomenout následující zdroj informací, a to internetovou stránku stackoverflow.com. Tato stránka byla vytvořena v roce 2008 Jeffem Atwoodem a Joelem Spolským. Jedná se o webový portál s otázkami a odpověďmi pro profesionální, ale i začínající programátory, který funguje na principu vkládání dotazů, kdy se ostatní členové diskusního vlákna a komunity snaží dotazujícímu pomoci. Ovšem ne všichni znají správnou odpověď. Pokud je vloženo více doporučení od různých autorů, tak ostatní mohou svými hlasy pomáhat správné odpovědi, aby se v zásobníkovém uspořádání dostala výše. Pokud tedy někdo má stejný nebo obdobný problém, najde tento dotaz a zjistí i správnou odpověď. Autor správné odpovědi také dostává jakési body, a tím si zlepšuje reputaci. Je pravda, že na této stránce jsou vkládány dotazy ohledně různých programovacích jazyků, a proto je nutné každé nové otázce přiřadit relevantní tagy. Od ledna 2019 má Stack Overflow přes deset milionů registrovaných uživatelů.

Fungování portálu lze představit na nejrůznějších obecných příkladech. Pokud má začínající programátor se slabší znalostí angličtiny obavy z pochopení svého dotazu, najdou se i tací přispěvatelé, kteří jeho formulace upraví (někdy i během velmi krátké doby (v rozsahu minut a hodin) do přijatelnější podoby anglického jazyka a za další krátký čas lze očekávat první odpověď s řešením původně neobratně formulovaného problému. Pokud by odpověď nebyla dostatečně přesná, stačilo by příspěvek okomentovat a počkat, až se jeho autor k příspěvku vrátí a řešení upřesní.

5 PŘÍKLADY APLIKACÍ

Pro vytvoření elektronického výukového kurzu bylo nutné nejdříve stanovit, pro koho bude kurz určen, a podle toho k vytváření kurzu přistupovat. Cílová skupina byla po úvaze zvolena, a to pokročilí studenti středních a vysokých škol. Vstupním předpokladem je znalost jazyka Python, i když jedna část kurzu se přímo zabývá možnostmi, jak si jazyk Python zopakovat. Lze totiž předpokládat, že ne všichni studenti, kteří se pokusí kurz zvládnout, budou mít dostatečné vstupní znalosti. Pokud tedy pro ně kurz bude obtížný, mohou se k doporučeným materiálům vrátit a ještě si základy jazyka Python zopakovat.

První možnost, jak si zopakovat nebo osvojit základy jazyka Python, je nabízen v podobě kurzu, který lze doinstalovat pomocí rozšiřujícího modulu přímo do prostředí PyCharm. Při spuštění kurzu je jeho účastník dotázán na roli, kterou zaujímá ve vzdělávacím procesu (učitel nebo student). Volba možnosti učitel umožňuje sledovat studenty a jejich znalosti ve vlastním kurzu. Celý kurz je rozdělen do deseti základních oblastí (úvod, proměnné, řetězce, datové struktury, podmínky, cykly, funkce, třídy a objekty, moduly a balíčky, vstup/výstup ze/do souboru). Každá oblast má různý počet cvičení. Cvičení má vždy svoji praktickou část, ve které je obsažen zdrojový kód, který musí být pro úspěšné splnění upraven či doplněn podle zadání. Pokud student usoudí, že kód programu správně dokončil, ověří si správnost pomocí stisknutí příslušného tlačítka pro vyhodnocení. Pokud je vše v pořádku, tak můžeme postoupit k dalšímu úkolu. Při zadání nesprávného řešení jsme na něj upozorněni a měli bychom cvičení zkusit znovu, případně je možné si správné řešení zobrazit. Jednou z výhod tohoto kurzu je, že se žák naučí využívat i možnosti PyCharmu, jako je kontextová nápověda, nebo jak pracovat s ladicími nástroji.

Druhá možnost je využití mobilní aplikace SoloLearn. V dnešní době používáme mobilní zařízení velmi často k různým účelům. Proč si tedy čas strávený například cestou do školy ve vlaku, autobusu nebo tramvaji nezkrátit něčím užitečným? Aplikace je volně dostupná jak v obchodě Google Play, tak na App Store. V současné době si v aplikaci můžeme vybrat z více než patnácti kurzů, ať už programovacích, skriptovacích nebo značkovacích. V aplikaci je kurz zaměřen přímo na poslední verzi Pythonu, tedy na verzi tři. Aplikace samotná je velmi jednoduchá a přehledná. Kurz je vždy rozdělen do několika oblastí (základní koncepty, kontrolní struktury, funkce a moduly, výjimky a soubory, funkční programování, objektově orientované programování, regulární výrazy a balíčky). Každá

oblast vždy zobrazuje nejdříve teorii a následuje úkol s ní spojený (někdy otázka, častěji ale doplnění kódu). Velkou výhodou je také to, že každý úkol má své vlastní vlákno s diskuzí k danému úkolu, pokud tedy nerozumíme úkolu nebo nevíme, jak úkol vyřešit, nahlédneme do diskuze, kde můžeme potřebné informace zjistit. Pokud si myslíme, že danou oblast známe, můžeme zkusit závěrečný kvíz. Pokud kvíz úspěšně absolvujeme, otevře se nám následující oblast a můžeme pokračovat dále.

Další možností je kniha od Marka Summerfielda Python 3, kterou vydalo nakladatelství Computer Press v roce 2013. Největší plus této knihy je, že je kompletně přeložena do českého jazyka. Cena knihy se pohybuje okolo 600 Kč. Rozhodně ale knihu najdete i v mnoha knihovnách. Má přibližně šest set stran, ale samozřejmě je ochuzena o interaktivní cvičení. Z tohoto důvodu ji lze považovat za kvalitní materiál pro získání určitého základu, ale pro opakování ji shledávám jako nejhorší možnost z uvedených, a to zejména z časového hlediska.

Poslední možností uvedenou v kurzu je tutoriál od youtubera jménem Telusko. Telusko vytváří tutoriály jak pro začátečníky, tak pro pokročilé programátory, například sérii videí zaměřených na Python, Javu, GIT, kotlin a mnoho dalšího. Jeho tutoriál zaměřený na Python je jeho zatím poslední série, která aktuálně čítá 81 videí. Videa jsou velmi dobře zpracovaná a ihned zaujmou. Youtuber má velmi dobrou výslovnost a slova jsou podpořena efekty, kdy se kolem něj objevují podstatné informace. Nevýhodou videí je, že sledující videa jen přijímá, ale sám si vytvoření programu nevyzkouší.

5.1 ŠABLONY ROZVRŽENÍ (LAYOUT'S)

Šablony slouží k rozvržení prvků ve svém prostoru. Ne vždy se nám hodí stejný způsob rozvržení, a proto si dále uvedeme základní informace o každém z nich. Je třeba si uvědomit, že prvkem layoutu může být další, a i jiný layout.

- AnchorLayout

Jak už může být patrné z názvu, jedná se o layout, kde vložený prvek bude někam v layoutu ukotven. Za tímto účelem nám slouží dva základní parametry a to anchor_x a anchor_y. Oba parametry mohou nabývat hodnot top (nahoře), left (vlevo), center (uprostřed), right (vpravo) a bottom (dole). Lze tedy odvodit, že tento layout můžeme připodobnit tabulce o výšce i šířce tří polí.

- BoxLayout

Prvky, které se přidávají do tohoto layoutu, se postupně řádí pod nebo vedle sebe. To je ovlivněno parametrem orientation, který může nabývat hodnot vertical nebo horizontal. Pokud do tohoto layoutu umístíme pouze jeden prvek, tak se jeho velikost přizpůsobí a zobrazí se přes celý layout. Pokud přidáme další prvek, velikost prvního se sníží tak, aby oba prvky byly stejně velké.

- FloatLayout

Layout žádným způsobem neurčuje, jak budou prvky rozmístěny, ani jak budou velké. Proto je nutno tyto parametry nastavit, slouží k tomu atributy size (velikost) a pos (pozice). Hodnoty můžeme zadávat ve velikosti pixelů, nebo využít desetinná čísla, kde je poté velikost brána jako procenta.

- RelativeLayout

Tento layout je velmi podobný předešlému, ale pozice je zde relativní vůči layoutu. V předchozím případě to bylo relativní vůči oknu, zde vůči layoutu.

- GridLayout

GridLayout slouží pro zarovnání prvků do mřížky, klíčovými atributy jsou cols (sloupce) a rows (řádky). Defaultně se nastavují všechny prvky stejně široké a vysoké. Pokud to chceme změnit, tak musíme využít atribut size_hint_x=None, tím zrušíme výchozí hodnotu a poté nastavíme šířku na naši velikost – width = 200. Pokud šířku nastavíme prvkům v jednom sloupci, tak se ty v druhém sloupci roztáhnou, aby nevznikl prázdný prostor.

- PageLayout

Layout, který má více stránek a napodobuje tedy listování knihou. Překlápění mezi stránkami je umožněno pomocí ohraničení.

- ScatterLayout

Chová se velmi podobně jako RelativeLayout, ale prvky lze volně přemístit, otočit a zvětšit/zmenšit pomocí dotyků nebo kliknutí.

- StackLayout

Princip je velmi podobný BoxLayout, prvky lze řadit osmi způsoby a klíčovým atributem je orientation. Pokud zadáme hodnotu rl-bt (right to left and from bottom to top), tak se

prvky budou řadit zprava doleva a od spodu nahoru. Velikost prvků v tomto případě nemusí být jednotná.[20]

5.2 PRVNÍ APLIKACE

Jako úplně první aplikace byla vybrána aplikace Hello World, se kterou se již každý určitě setkal. Cílem aplikace je vytvořit okno a v něm text. Za tímto účelem musíme vytvořit třídu, která bude potomkem třídy App. Při jejím vytváření musíme metodě build nastavit návratovou hodnotu na prvek Label s požadovaným textem. Následně zadáme příkaz pro spuštění třídy. Pro funkčnost aplikace jsou nutné vhodné importy z knihovny Kivy. Výsledek by měl vypadat takto.

```
from kivy.app import App  #Importování z knihovny Kivy pro vytvoření aplikace
from kivy.uix.label import Label #Importování z Kivy pro grafický prvek label

class TestApp(App):                    #Vytvoření třídy typu App
def build(self):                       #Funkce pro vytvoření třídy
return Label(text='Hello World')       #Nastavení výstupu okna

TestApp().run()                        #Kód pro spuštění třídy
```

5.3 JEDNODUCHÉ APLIKACE

Úkoly v elektronickém kurzu mají za cíl postupně a přiměřeně rozvíjet a prohlubovat znalosti studenta. Příklady byly samostatně vymýšleny, případně inspirovány předchozím studiem či internetovými zdroji. Úkol vždy začíná úvodním textem, kde student zjistí, na co je daná aplikace zaměřena a jaký je její cíl. Dále je student upozorněn na to, co bude potřebovat a je částečně seznámen s novinkami, které bude muset využívat. V podstatě se jedná o jednoduchý návod. Samozřejmě může student úkol vytvořit zcela podle vlastní úvahy, což vede k hlubšímu pochopení a většímu zapojení kreativity, která je v programování zásadní. Součástí úkolu je i ukázkové řešení, které se zobrazí až po kliknutí na odkaz. Student je upozorněn, že pokud si výsledně řešení prohlédne, nemůže dojít k úplnému pochopení problematiky. Může se ovšem stát, že některý student nebude schopen řešení sám vymyslet, proto každý jednodušší program obsahuje návrh na vylepšení. Pokud si tedy výsledek zkopíruje, vyzkouší a následně provede navržené změny, mělo by k určitému porozumění problematiky dojít.

[20]Layouts. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/gettingstarted/layouts.html

5.3.1 UKÁZKA JEDNODUCHÉ APLIKACE – KRESLENÍ

Tento příklad je zaměřen na kreslení, které se velmi často ve školách používá pro svoji názornost. Pokud jste se s tím v klasickém prostředí Pythonu nebo jiného jazyka setkali, tak by pro vás neměl být úkol příliš složitý. Některé znalosti si zopakujete a v oblasti vytváření Android aplikací si rozšíříte obzory. Úkolem aplikace je po stisknutí vytvořit na plátno nějaký obrazec a tahem prstu tak budeme kreslit čáru.

Co budeme pro tuto aplikaci potřebovat? Základní aplikaci si můžeme rozdělit na dvě části pomocí layoutu (GridLayout). V jedné části bude umístěno tlačítko pro vyčištění plátna a v druhé části plátno.

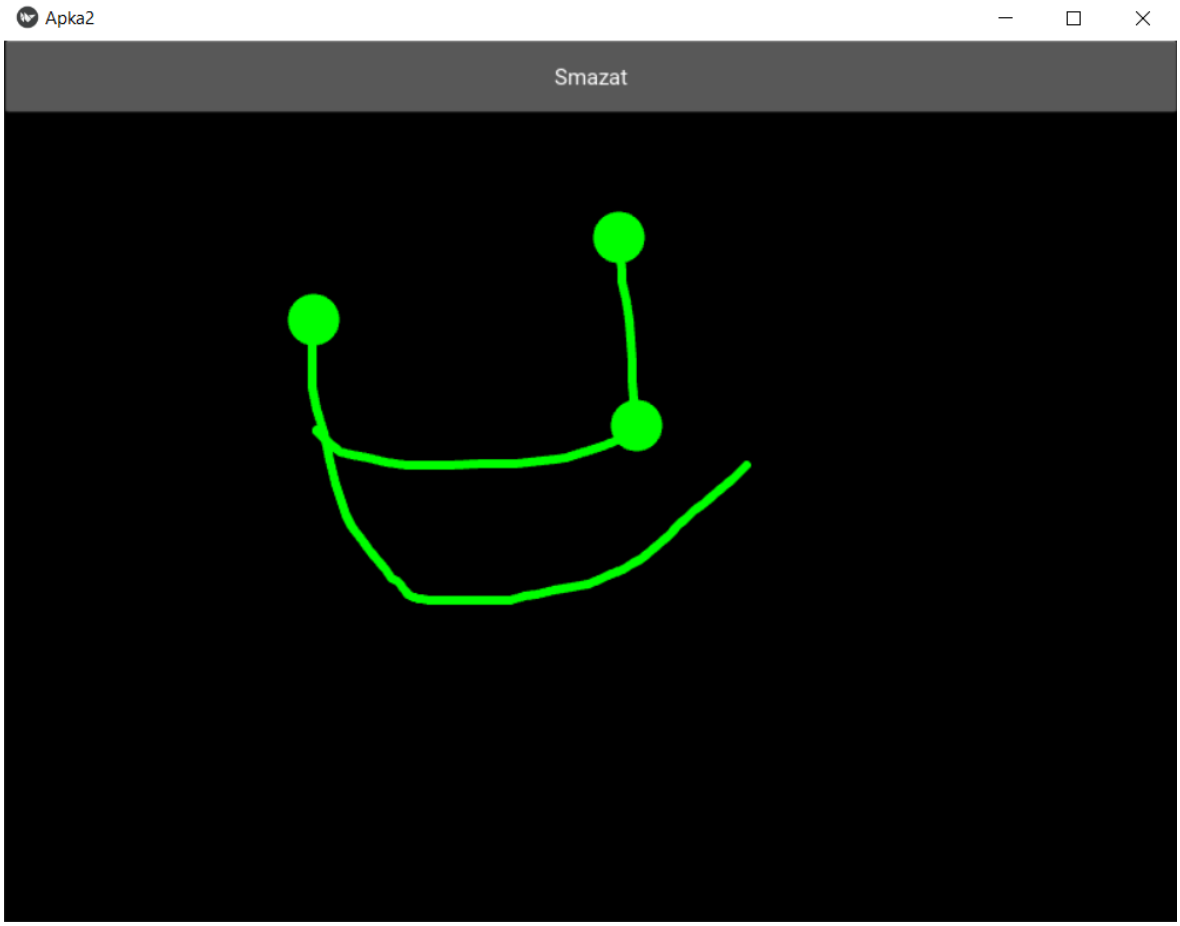
Pro práci s grafikou budeme potřebovat části knihovny Kivy, které musíme importovat.

```
from kivy.uix.widget import Widget
from kivy.graphics import Ellipse, Line
```

Pro snadnou práci bude nejlepší plátno vytvořit jako vlastní třídu a definovat v ní funkce pro kreslení.

```
def on_touch_down(self, touch):
with self.canvas:
def on_touch_move(self, touch):
```

V hlavní třídě tedy definujeme rozložení aplikace, vložíme prvky a vytvoříme funkci pro smazání obsahu plátna. Pokud si nebudete vědět rady, tak na další stránce je ukázáno možné řešení. (Je třeba ukázkové řešení brát jako pomůcku, a ne si prohlédnout kód a myslet si, že došlo k porozumění!)



Obrázek 8: Ukázka možného výsledného řešení (zdroj: vlastní)

Pokud se Vám povedlo program vytvořit, tak si zkuste změnit velikosti vašeho kresleného objektu, šířku a barvu čáry, změnit výchozí velikost tlačítka atp.

5.3.2 POPIS VYTVÁŘENÍ ZDROJOVÉHO KÓDU APLIKACE

Nejdříve si vytvoříme nový projekt a zvolíme jeho adekvátní název. Dále vytvoříme nový soubor s koncovkou .py a vhodným názvem, který se bude ukazovat jako název rámce. Ve vytvořeném souboru nejdříve vytvoříme hlavní třídu pomocí slovíčka class, její název by se měl shodovat s názvem souboru. Za názvem vytvoříme kulaté závorky a zadáme parametr App, který určuje, že se jedná o aplikaci. V tuto chvíli jsme upozorněni, že se jedná o nevyřešenou referenci, a tudíž musím na úplný začátek přidat importování třídy App. Importování provádíme slovíčkem from a import. Po slovíčku from definujeme, odkud chceme importovat, následuje slovíčko import, zde určíme, kterou část z knihovny potřebujeme. Po uzavření závorky následuje dvojtečka a můžeme definovat její obsah. Nyní si vytvoříme metodu pomocí slovíčka def, která bude definovat vzhled a obsah aplikace. Metoda se bude jmenovat build jako sestavování a její parametr bude self, aby

bylo označení objektu jednoznačné. Dále musíme objektu přidat kreslicí plátno typu MyPaintWidget, které si později definujeme jako vlastní třídu. Nyní využijeme šablony pro rozvržení aplikace. V tomto místě definujeme layout a využijeme GridLayout se dvěma řádky. Na další řádce si vytvoříme proměnnou typu tlačítko, které nastavíme parametr text, zrušíme výchozí výšku tlačítka, protože nechceme, aby nám zabíralo polovinu prostoru, a nastavíme výšku, která je pro nás vhodná. Dále na tlačítko navážeme funkci, která bude reagovat na uvolnění tlačítka. Pokud uvolníme tlačítko, bude vyvolána metoda objektu pro vymazání plátna, kterou si zanedlouho definujeme. Tlačítko i plátno máme připraveno, tudíž ho můžeme přidat do layoutu a celá tato metoda bude mít tento layout jako návratovou hodnotu. Samozřejmostí je, že musíme všechny grafické prvky importovat. Nyní vytvoříme metodu pro smazání plátna, opět tedy pomocí slovíčka def a jména metody, kde budou parametry self a objekt. Pro smazání plátna stačí pomocí slovíčka self, názvu plátna a definice plátna zavolat metodu clear. V tuto chvíli již jen stačí definovat konstrukci, která se zavolá a spustí hlavní třídu. Využijeme podmínky if a pokud __name__ odpovídá hodnotě__main__, tak se zavolá hlavní třída a metoda run z knihovny Kivy.

Aktuálně tedy máme vyřešené spouštění aplikace a hlavní třídu. Zbývá již jen vytvořit plátno, na něž se bude kreslit. Plátno vytvoříme jako celou třídu, která bude založena na prvku Widget. Po definování hlavičky třídy tedy vytvoříme nejdříve metodu, která bude reagovat na stisknutí. Metoda musí nést název on_touch_down s parametry self a touch, abychom věděli, kde došlo k dotyku. V zadání bylo určeno, že po stisknutí se má zobrazit nějaký obrazec, v tomto případě vyzkoušíme vytvořit na daném místě kruh. Uvnitř metody využijeme slovíčka with, které se používá při práci s nespravovanými prostředky. V našem případě potřebujeme pracovat s vlastním plátnem. Nastavíme si barvu pera, vytvoříme proměnnou d, která bude určovat poloměr pro všechny vytvořené kruhy. Zavoláme metodu pro vytvoření elipsy z knihovny kivy.graphics a nastavíme pozici podle místa dotyku a velikost v ose x a y, která je u kruhu stejná. Pokud bychom aplikaci otestovali, tak bychom zjistili, že při stisknutí a následném pohybu aplikace není dobře ošetřena a dojde k jejímu pádu, proto musíme definovat reakci na tuto událost. Proto doplníme metodu pro tažení, která bude vytvářet čáru. Čára bude mít ovšem jen jeden bod, a tudíž nebude vidět, jde jen o ošetření pádu aplikace. Pro opravdové kreslení si

definujeme metodu on_touch_move, která bude mít opět parametr self a dotyk. Nyní tedy budeme kreslit čáru a je nutné neustále načítat místo dotyku pro vytvoření čáry. Parametr dotyku vždy obsahuje x a y souřadnici. Postupně při změnách souřadnic dotyku kreslíme čáru, která se prodlužuje a navazuje. Pokud požadujeme funkčnost aplikace, nesmíme zapomenout importovat prvky Color, Ellipse a Line. Výsledek je k nahlédnutí v příloze.

5.3.3 JEDNODUCHÉ APLIKACE – OSTATNÍ ÚKOLY

Další aplikace byla zaměřená na načtení textu. Cílem bylo vypsat zadaný text pozpátku. Poté následovala aplikace, kdy se při dotyku tlačítka zobrazil libovolný obrazec, a pokud došlo k tažení, tak se vykreslovala čára. Následující aplikace sloužila jako prohlížeč souborů, kde se uživatel mohl pohybovat adresářovou strukturou pomocí prvku FileChooserListView. Jedna z navržených aplikací sloužila pro jednoduché násobení matic o velikosti 2x2. Poslední aplikace v této sekci byla zaměřena na použití knihovny psutil. Jejím úkolem bylo vypisovat informace o daném zařízení.

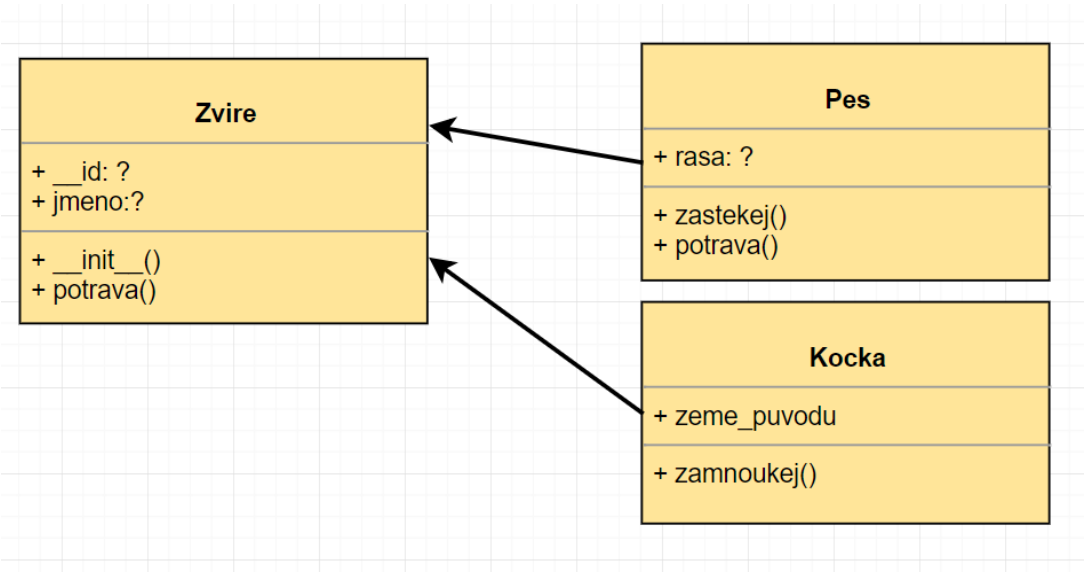
Následující sekce byla věnována prohloubení znalostí a příklady byly založeny na těch předchozích. Zde se tedy nacházel úkol pro vytvoření aplikace, kde si v menu uživatel vybere tvar a podle výběru může daný tvar umisťovat na plátno, dále pomocí tažení může kreslit čáru. Součástí menu je jako v původní aplikaci tlačítko pro smazání plátna.

Druhým úkolem v této sekci bylo navázání na procházení adresářovou strukturou. Cílem aplikace bylo načtení dat z příslušného souboru a následně z hodnot v souboru vypočítat kvadratickou rovnici.

5.4 NÁVRH POSTUPU VYTVÁŘENÍ APLIKACÍ

Vývoj aplikací nezačíná tím, že rovnou začneme psát kód aplikace. Prvním krokem by mělo být zjištění, co vše aplikace musí umět a jak by měla vypadat. Dále také záleží na tom, zda bude aplikaci vyvíjet jeden programátor, nebo celý tým. Pokud má programátor celkový obraz o aplikaci, tak by mohlo být prvním krokem sestavení UML diagramu. Diagram dobře ukazuje různé vazby v programu a mezi soubory (konkrétně v jazyce Python mezi třídami), do diagramu se také umisťují proměnné dané třídy a její metody. Jinými slovy jde o zjednodušený návrh celé aplikace. Pokud tedy danou aplikaci vytváří tým

programátorů, musí být zajištěna dělba práce. UML diagram zajistí i konzistenci názvů proměnných a metod.



Obrázek 9: UML diagram (zdroj: vlastní)

5.4.1 VLASTNÍ NÁVRH POSTUPU VÝVOJE ANDROID APLIKACÍ

Samotný vývoj aplikace může být rozdělen do pěti etap. V první etapě je nutné **zjistit všechny potřebné informace** o výsledné aplikaci. Pokud má programátor kompletní informace, získá tak určitou představu, jak by mohla výsledná aplikace vypadat. Klíčové informace jsou, co vše má finální aplikace umět. Je také nutné zjistit, jestli nám všechny vstupní informace budou dodány, nebo jestli je budeme muset získat například z databáze. Pokud ano, musí zákazník, případně programátor získat přístup k těmto datům. Pro jednodušší vývoj může být vytvořen algoritmus aplikace. Zásadní informací může být, pro která zařízení bude aplikace určena. Pokud bude aplikace nasazena na nějaký konkrétní model mobilního zařízení, tak může výsledek vypadat lépe.

Druhým krokem by mohlo být **vytvoření uživatelského rozhraní aplikace**. V tuto chvíli už by mělo být jasné, které komponenty budeme potřebovat. V ideálním případě je možné udělat návrh grafického prostředí a ten následně konzultovat se zákazníkem. Pro vytvoření grafického rozhraní je určitě vhodné využít možnosti Kivy Language, a tím oddělit grafickou část od funkčnosti. Výhodou je, že oba kódy nebudou narušeny a kódy tedy budou čistší a lépe čitelné. Programátor by měl také brát v úvahu to, aby byla aplikace uživatelsky přívětivá, tím je například myšleno, aby funkční prvky byly dostatečně

daleko od sebe, aby uživatel omylem nezměnil nějakou vstupní hodnotu. Ohroženými prvky jsou především prvky jako slider nebo checkbox.

Třetím krokem při vývoji aplikací je **psaní samotného kódu aplikace**. Programátor může aplikaci pro Android psát jak v Pythonu, tak v Javě. Zde záleží na zkušenostech a preferencích programátora. Samozřejmě pokud zákazník preferuje univerzální aplikaci, která bude fungovat jak na zařízeních s Androidem, tak i s iOS, je pro programátora lepší volbou Python a knihovna Kivy.

Předposledním krokem je **testování aplikace**. Nejdříve se aplikace testuje, zda pracuje tak, jak při očekávaných vstupech má. Pokud ano, tak by se měl tester dále zaměřit na možné způsoby, jak aplikaci přinutit k pádu. Často se v praxi setkáváme s tím, že aplikace nemá dostatečně ošetřené vstupy. Například pokud místo číselné hodnoty zadáme text. Tyto vstupy ošetřujeme pomocí výjimek, abychom mohli na nepřípustný vstup vhodně reagovat. Například vyskakovacím oknem s adekvátním popsáním nastalé situace.

Posledním krokem je **zohlednění a náprava zjištěných chyb** při testování. Po dokončení by aplikace měla být nasazena do testovacího provozu. Pokud se během testovacího provozu neobjeví žádné chyby, je možné ji považovat za dokončenou[21].

5.5 POSTUP PRO VYTVOŘENÍ ANDROID BALÍČKU

Vytvoření Android balíčku není napoprvé jednoduchá záležitost. Po prvotní instalaci a konfiguraci potřebných nástrojů bude ale postup již poměrně snadný. Postup bude následující: nejdříve musíme ověřit, zda je na našem stroji povolená virtualizace. Pokud ne, musíme ji povolit. Následně je nutné nainstalovat Oracle VirtualBox a v něm spustit speciálně upravenou verzi operačního systému Ubuntu. Dalším krokem je vytvoření sdílené složky mezi naším strojem a strojem virtuálním. Předposledním krokem je úprava souboru, který ovlivňuje výsledný balíček. Nakonec je nutno v terminálu spustit příkaz, který nám z dané aplikace vytvoří Android balíček.

Zda máme povolenou virtualizaci, zjistíme ve Windows pomocí správce úloh. Nejsnadněji se do správce úloh dostaneme klávesovou kombinací Ctrl+Shift+Esc. Pokud se zobrazí jen

[21]HOFMAN, Martin. *Vývoj Android aplikací v prostředí IntelliJ*. Plzeň, 2017. Bakalářská práce. Západočeská univerzita v Plzni.

základní okno, klikneme ve spodní části na "Více informací". Dále otevřeme záložku Výkon a v části pro procesor pod grafem zjistíme, zda je virtualizace povolena.

Virtualizaci povolíme ve Windows 10 následujícím způsobem. Otevřeme nastavení, zde položku Aktualizace a zabezpečení. V levém panelu nalezneme položku Obnovení, zde nalezneme Spuštění s upřesněným nastavením a pod ním stiskneme tlačítko Restartovat Hned. Po restartování stroje se zobrazí obrazovka s výběrem, kde zvolíme Řešení problémů, dále Pokročilé možnosti a poté Nastavení UEFI firmware, kde potvrdíme restartování stroje. Nyní by se mělo zobrazit startovací menu s možností BIOS Setup, kterou zvolíme, a tím se dostaneme do nastavení BIOSu. V BIOSu se přesuneme do záložky Systémová konfigurace a zde povolíme virtualizaci. Následně uložíme změny a opustíme BIOS.

V tuto chvíli je stroj připravený na instalaci softwaru Oracle VM VirtualBox, který získáme přímo na stránkách Oracle. Ze stránky rovnou získáme i Oracle VM VirtualBox Extension pack. Pomocí průvodce nainstalujeme VirtualBox. Průvodce se zeptá, kam má být program nainstalován, zda chceme vytvořit zástupce na plochu a jestli má provést asociaci s příslušnými soubory. V tuto chvíli se dostaneme k příslušné instalaci, kterou spustíme. Během instalace se objeví dotaz, zda chceme instalovat ovladače pro práci s USB, kde zvolíme možnost instalovat.

Pro potřebnou funkčnost VirtualBoxu spustíme po jeho nainstalování i instalaci Oracle VM VirtualBox Extension pack. Před zahájením instalace musíme potvrdit licenční podmínky a následně se instalace během několika desítek sekund provede.

Upravenou verzi Ubuntu najdeme přímo na stránkách Kivy.org pod označením Kivy Buildozer. Z webových stránek ji získáme, rozbalíme a spustíme VirtualBox. Zde přes záložku Soubor a Importovat applienci vybereme rozbalenou verzi Kivy Buildozer.

Po úspěšném importování je nutné otevřít nastavení applience a upravit následující parametry:

Obecné –> Pokročilé parametry –> Sdílené schránky a Táhni a pusť nastavit na hodnotu Obousměrné. Dále ještě Sdílené složky –> Přidat novou a vybrat složku s projekty.

Momentálně je applience připravena a můžeme ji spustit. Po spuštění je vyžadováno heslo, které je „kivy“ (pozor na anglickou klávesnici). Dalším krokem je zkopírování

projektu ze sdílené složky například do složky Home. Po otevření složky s projektem pomocí pravého tlačítka myši vybereme Open Terminal Here. V terminálu napíšeme přikaz „buildozer init“, který nám ve složce vygeneruje soubor pro nastavení výsledné aplikace. Soubor otevřeme a můžeme změnit název výsledného balíčku atp. Podstatné pro nás je změnit „log_level“ na 2 a soubor uložit. Nyní se vrátíme do terminálu a použijeme příkaz „buildozer -v android debug“ a ten nám vytvoří z dané aplikace apk balíček. (Napoprvé může tato operace trvat několik minut. Pokud se operace nezdaří, je nutné postupovat podle poslední verze Buildozer dokumentace.) V tuto chvíli je balíček hotov a můžeme ho přesunout do sdílené složky a spustit si ho přímo ve svém Android zařízení nebo ve virtuálním prostředí Androidu. Velmi povedené virtuální prostředí Androidu je Nox. Nox je emulátor pro Android, který je dostupný zdarma, a to jak pro Windows, tak pro MacOS na adrese bignox.com.

5.6 DYNAMICKÉ POZICOVÁNÍ

Při vytváření Android aplikací existují dva způsoby, jak k vytváření grafického uživatelského prostředí přistupovat. První možnost je vytváření grafického prostředí přímo v Python souboru, což se může zdát jako jednoduché řešení, ovšem pokud se bude jednat o složitější aplikaci, tak nám definice uživatelského prostředí bude kód nejen prodlužovat, ale částečně i narušovat. Za tímto účelem byl vytvořen Kivy language. Jednoduše stačí vytvořit soubor se stejným názvem jako má hlavní soubor s koncovkou .py a nastavit mu koncovku na .kv. V KV souboru tedy následně můžeme definovat uživatelské prostředí zcela odděleně. Takový soubor by se dal přirovnat k CSS souborům u webových stránek[22].

Pokud v hlavní třídě, která je založena na třídě App, zavoláme vlastní návrh třídy, která je typu Widget, tak stačí této třídě nastavit jedinou hodnotu, a to hodnotu pass. Slovíčko pass se používá, pokud nechceme třídu pomocí kódu v Pythonu nijak konkrétně definovat. Někdy se slovíčko pass používá v případě, že přerušíme práci a poté víme, že by zde měl být nějaký zdrojový kód. V našem případě nám slovíčko pass postačuje a k upravení grafického prostředí třídy dojde v KV souboru pomocí Kivy language. Do .kv souborů můžeme importovat Python knihovny, které se budou chovat úplně stejně jako v našem .py souboru. Je nutné si také uvědomit, že pokud umisťujeme prvky přímo do

[22]Kivy Language. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/api-kivy.lang.html

Widgetu, tak nulový bod v prostoru je v levém dolním rohu, proto musíme relevantně nastavit pozice prvků.

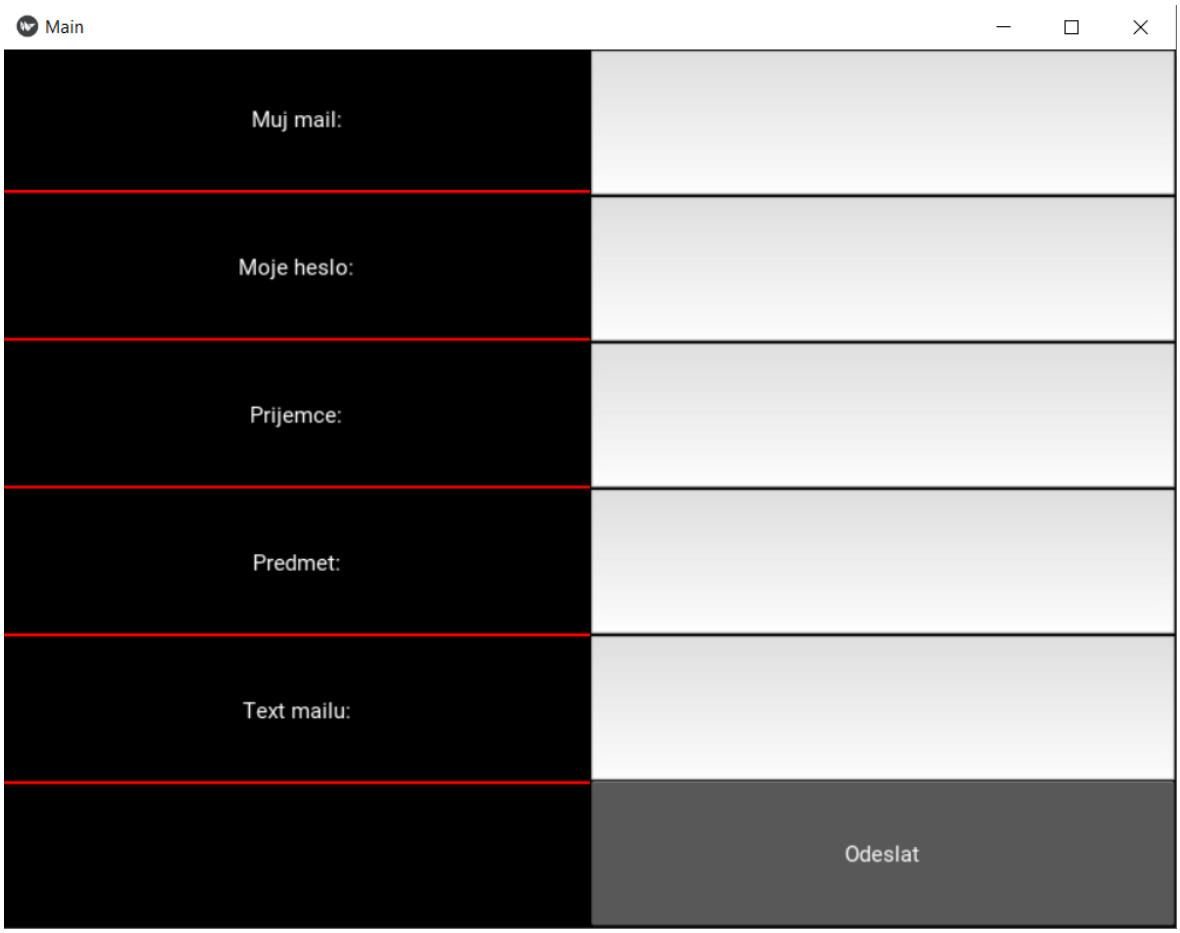
5.6.1 ÚKOL NA PROCVIČENÍ KIVY LANGUAGE

Součástí elektronického kurzu je také úkol, který je přímo zaměřen na procvičení jazyka Kivy. Úkol zaměřený pouze na grafickou stránku není vhodný, a proto byla do úkolu zapojena další technika, se kterou se programátor v běžné praxi setkává. Jedná se o techniku, kdy programátor musí pracovat se zdroji informací. Ne vždy je vhodné hledat vše v dokumentaci, ale je možno využít článků na internetu.

5.6.2 KONKRÉTNÍ ÚKOL NA PROCVIČENÍ KIVY LANGUAGE – ODESLÁNÍ MAILU

Tento příklad bude trochu odlišný, ale bude se blížit běžné praxi programátora. Programátor si nemůže pamatovat veškeré funkce a principy, proto musí umět pracovat se zdroji. V dnešní době je hlavním zdrojem informací internet. My budeme mít tu výhodu, že budeme mít hned jeden kvalitní zdroj doporučený.

Úkol: Vytvořte aplikaci, která bude z vašeho Google mailu odesílat mail někomu jinému. Jako primární zdroj informací využijte https://realpython.com/python-send-email/. Nejsnazším způsobem jak oddělit prvky, pokud nejsou tvořeny graficky, je vložení widgetu, který je tvořen čárou.



Obrázek 10: Ukázka možného výsledného řešení (zdroj: vlastní)

5.6.3 POPIS VYTVÁŘENÍ ZDROJOVÉHO KÓDU APLIKACE

Pokud je cílem odeslat mail, je podstatné si uvědomit, jaké informace budeme potřebovat, a podle toho uživatelské prostředí uspořádat. Většinou je popisek nad nebo vedle okénka pro vstup. V našem případě si zkusíme variantu, kdy je popisek vedle vstupu. Budeme potřebovat svůj email, heslo ke svému účtu, cílovou adresu, předmět a text emailu. Dále musíme mít tlačítko pro odeslání emailu. Pro návrh jsem tedy zvolil BoxLayout, který bude orientován horizontálně a bude obsahovat další dva BoyLayouty, které jsou orientovány horizontálně. Tudíž mohu přidávat prvky do šablon pod sebe. První šablona bude obsahovat popisky, které budeme chtít oddělovat, a druhé vstupní pole a na závěr tlačítko. Nejsnazším způsobem, jak oddělit prvky typu Label od sebe, je definování vlastního widgetu, který bude vykreslovat obdélník. Nejdříve si vytvoříme soubor s koncovkou .kv, kde nejdříve uvedeme, ke které třídě typu Screen se bude vázat. Jelikož bude mít aplikace pouze jednu obrazovku, můžeme si obrazovku nazvat například MainScreen. Následně si vytvoříme speciální widget pro oddělování popisků. Nastavíme

mu id, zrušíme výchozí výšku, nastavíme vlastní. Následně přes Canvas nastavíme barvu, vykreslíme obdélník, kterému určíme pozici a velikost. Výhodou takto přednastaveného prvku je, že dále již budeme samotný prvek jen volat a nemusíme ho znovu definovat. Následně se tedy pustíme do použití šablony. Jako první přidáme na obrazovku BoxLayout a uvedeme jeho orientaci. Dále vložíme postupně popisky, které budou proloženy widgety pro oddělení. Popisky budou mít jedinou vlastnost, a to vhodně nastavený parametr text. Po dokončení prvního sloupce přidáme druhý BoxLayout, opět orientovaný vertikálně, a postupně přidáme okénka pro vstup a na závěr tlačítko. Každý vstup bude mít vhodně nastavené id, abychom mohli dané informace předat metodě pro odeslání emailu. Jelikož požadujeme po uživateli i heslo, tak danému vstupu nastavíme parametr password na hodnotu True. To způsobí, že se v políčku budou objevovat hvězdičky místo znaků. Posledním úkolem je zareagovat na uvolnění tlačítka, a tedy zavolat metodu třídy pro poslání emailu se zadanými vstupy.

Grafické prostředí již máme hotové a teď se pustíme do vytvoření Python kódu. První v souboru vytvoříme třídu, která bude založena na třídě App z knihovny Kivy. Tato třída bude obsahovat metodu build s parametrem self a její návratovou hodnotou bude proměnná vhodného názvu. Mimo třídu tuto proměnnou definujeme a přiřadíme ji spuštění souboru pomocí třídy Builder z knihovny kivy.lang s naším grafickým návrhem. Dále definujeme podmínku pro spuštění aplikace, pokud tedy __name__ odpovídá hodnotě __main__, tak se zavolá hlavní třída založená na třídě App.

Dalším krokem je vytvoření třídy, která je založena na třídě Screen z kivy.uix.screenmanager. Definujeme tedy třídu a v ní deklarujeme naši jedinou metodu pro odeslání emailu. Metoda bude mít jako vždy parametr self a dále zadané hodnoty z formuláře. V samotné metodě si definujeme proměnné, například proměnnou port nastavíme na hodnotu 587, kde probíhají služby smtp a nastavíme proměnnou smtp_server na hodnotu serveru, v našem případě na smtp.gmail.com. V tuto chvíli vytvoříme proměnnou zpráva, jejíž součástí bude předmět a text zprávy. Aby se předmět správně zobrazil v emailu, musí mít zpráva následující formát: “Subject: {}\n\n{}“.format(předmět, text zprávy). Následně vytvoříme context, který vytvoří ssl připojení. Je nutné knihovnu ssl importovat na začátku souboru, zároveň můžeme importovat knihovnu smtplib, kterou budeme využívat vzápětí. Nyní pomocí klíčového

slova with a knihovny smtplib definujeme smtp server s hodnotami název smtp serveru a portem. V tuto chvíli zavoláme server a metodu pro nastartování TLS a parametrem je vytvořený context. Nyní serveru předáme údaje pro přihlášení k emailové schránce a na závěr zavoláme metodu serveru sendmail, která má parametry vlastí emailové adresy, cílové emailové adresy a zprávy (která obsahuje předmět a text emailu). Výsledek je k nahlédnutí v příloze.

6 PRINCIPY OOP

Objektově orientované programování je princip programování, ale v jazyce Python se jeho implementace od jazyka Java značně odlišuje. Python nám umožňuje programovat procedurálně, objektově, funkcionálně, ale i v libovolné kombinaci předchozího.

Pokud se chystáme sestavit nějaký program, který není příliš rozsáhlý, můžeme jej snadno naprogramovat procedurálně, ale pokud se jedná o nějaký rozsáhlejší projekt, pak přináší objektově orientovaný přístup mnoho výhod.

Pokud se tedy zaměříme na procedurální styl, měli jsme v podstatě sekvenci příkazů, které se vykonávaly řádku po řádce. OOP je v dnešní době nejvíce používaný přístup pro vývoj programů a aplikací. Tuto metodiku v současnosti podporuje většina programovacích jazyků. Určité zdroje také tvrdí, že by se tento přístup měl využívat vždy, ale ne vždy je přínosem.

Jednou z podstat OOP je princip znovupoužitelnosti. Pokud máme některé komponenty hotové, tak není důvod je vytvářet znovu. V automobilovém průmyslu také mnoho automobilek používá stejné motory a další části. Pokud tedy následně sestavujeme automobil, je jednodušší ho složit z hotových komponent, než se zabývat vývojem nových koncepcí, které nemusí vést k tak dobrému výsledku. Také je důležité si uvědomit, že stávající komponenty jsou již otestované! Pokud tedy program/auto dobře sestavíme a někde je chyba, stačí opravit jen jednu součástku/třídu.

Dříve jsme se snažili psát kód z pohledu, jak se na něj dívá počítač, ale dnes se již přesouváme k pohledu člověka. Základní jednotkou je samozřejmě objekt, kterým může být dopravní prostředek, osoba nebo databáze. Každý objekt má své atributy a metody. Atributy jsou vlastnosti objektů (například značka automobilu), v podstatě se jedná o běžné proměnné. Objekt má kromě atributů metody, což jsou možnosti, co může objekt vykonat (například automobil může jet). Je samozřejmé, že metody mohou mít různé parametry a mohou mít návratovou hodnotu[23].

[23]Lekce 1 - Úvod do objektově orientovaného programování v Pythonu. *ITnetwork.cz* [online]. [cit. 2019-04-08]. Dostupné z: https://www.itnetwork.cz/python/oop/python-tutorial-uvod-do-objektove-orientovaneho-programovani

6.1 ZÁKLADY OBJEKTOVĚ ORIENTOVANÉHO PROGRAMOVÁNÍ

Základem OOP je objekt, který má své atributy a chování. Jako objekt můžeme vzít například psa. Jeho atributy jsou jméno, rasa a věk. Chování neboli funkce jsou štěkání, běhání atp.

Základní pilíře OOP jsou:

- Dědičnost – proces, kdy můžeme upravit třídu, která je vázána na jinou bez úprav stávající třídy.
- Zapouzdření – skrytí soukromých částí objektů.
- Polymorfizmus – koncept použití stejné operace různými způsoby pro různé vstupy.

Pro vytváření objektů musíme mít návrh objektu, jeho návrhem je třída. Můžeme si tedy představit třídu jako náčrtek psa s popisky. Třída obsahuje údaje o jméně, rase a věku. Pokud tedy vytvoříme instanci třídy Pes, tak se stává objektem. Objekt je zároveň i instance. Když definujeme třídu, tak definujeme popis objektu. V tu chvíli není zatím přidělena žádná paměť. Při vytvoření konkrétní instance třídy Pes má už objekt své místo v paměti, pokud tedy třída obsahuje metodu __init__(). Dalším prvkem jsou metody. Metody jsou funkce definované uvnitř třídy. Používají se k definování chování objektu.

6.1.1 DĚDIČNOST

Dědičnost je způsob, jak na základě naší třídy vytvořit další, která je více specializovaná. Nová třída získá od předchůdce atributy a metody. Dále specializované třídě můžeme přidat další atributy a metody, které jsou pro ni specifické. Příkladem může být třída zvíře. Na jejím základě můžeme postavit třídu kočka, pes atp. Možnost vytvářet podtřídy je jednou z velkých výhod OOP, protože tak můžeme jednoduše vytvořit novou třídu, která je založena na již stávající vyzkoušené a otestované třídě, ke které doplníme potřebné atributy a metody.

Jestliže tedy máme novou zděděnou třídu, tak nám ale nemusí všechny její metody vyhovovat. Ovšem v podtřídě lze libovolné metody modifikovat. Pokud máme v původní a zděděné třídě stejnou metodu, objekt zděděné třídy správně zavolá tu metodu, která je ve zděděné třídě – jedná se o dynamickou vazbu. Pokud bychom ale chtěli volat tuto metodu z rodičovské třídy, tak k tomu slouží funkce super(). Když vytváříme vlastní třídy, měli bychom se držet konvence. Název třídy tedy začíná velkým písmenem.

V dalším objektově orientovaném jazyku jako například Java existuje princip přetěžování (ve stejné třídě máme metody se stejným názvem, ale s odlišným počtem nebo pořadím parametrů), který Python nezná. Ovšem to pro nás díky možnostem práce s argumenty v Pythonu není omezující.

6.1.2 ZAPOUZDŘENÍ

Zapouzdření nám slouží k zabalení dat a metod, ke kterým chceme znemožnit přímý přístup. Tato data a metody samozřejmě chceme používat, ale jen uvnitř třídy. Pokud tedy vytvoříme instanci naší třídy, můžeme ji přirovnat k černé skříňce. Pro práci s naší skříňkou nám slouží rozhraní, kterému zadáme požadavky nebo informace, a ona je zpracuje. Nemusíme vědět, jak skříňka funguje, ale víme, jak se k ní chovat. Chceme, aby uživatel mohl skříňku používat, ale zamezit jejímu případnému poškození.

Příklad: Pokud si vytvoříme třídu osoba, kde bude jedním z parametrů datum narození, bude podle toho ovlivněn atribut věk. Kdyby nám někdo zvenčí změnil parametr datum narození, tak by se hodnota atributu věk stala neplatnou. Došlo by tedy k tomu, že by byl objekt nekonzistentní. Tomuto chceme pomocí principu zapouzdření zamezit. Pokud vytvoříme objekt osoba, atribut označíme jako soukromý a nebude viditelný zvenčí. V případě, že bude datum špatně zadané, objekt nabídne metodu zmena_datum_narození(), která změní datum narození a zároveň aktualizuje obsah v proměnné věk.

6.1.3 POLYMORFIZMUS

Polymorfizmus je velmi úzce spjatý s dědičností. Polymorfizmus nám umožňuje používat jednotné rozhraní pro práci s různými typy objektů. Příkladem mohou být například písmena. Můžeme vytvořit třídu abeceda, která by obsahovala metodu pro vykreslení písmena. Pokud by všechna písmena dědila od abecedy, zdědila by její rozhraní. Je jasné, že každé písmeno vypadá jinak. Polymorfizmus nám umožňuje přepsat metodu pro vykreslení u každé podtřídy. Takže pokaždé budeme volat stejnou metodu, ale její výsledek bude záviset na typu objektu. Jednoduše řečeno jde o metodu, kterou mají všichni potomci definovanou se stejnou hlavičkou, ale jiným obsahem.

6.2 VYTVÁŘENÍ TŘÍD A OBJEKTŮ

Vytváření vlastních tříd je velmi jednoduchou záležitostí, je ovšem nutné si uvědomit několik podstatných informací a principů. Pokud vytváříme třídu, která je potomkem třídy jiné, tak získává její možnosti, které můžeme využívat. Ovšem pokud vytvoříme třídu a neuvedeme žádnou rodičovskou třídu, je vytvořená třída stejně potomkem. V tomto případě se třída stává potomkem třídy Object.

Třídy většinou obsahují několik metod, kdy jejím prvním parametrem musí být parametr self, což je odkaz ukazující na samotnou třídu. V Javě se obdobně používá slovíčko this. Slovíčko self se používá z důvodu absolutní srozumitelnosti.

Jednou z dalších odlišností Pythonu je, že pokud chceme vytvořit instanci, musíme provést dva kroky a nikoliv jeden, jak bývá zvykem. Python nejdříve vytvoří holý objekt, a až poté dojde k jeho inicializaci. Pro vytvoření instance nám slouží tzv. magická metoda __new__() a pro inicializaci objektu slouží metoda __init__(). Jestliže vytváříme objekt zděděné třídy, stačí upravit metodu pro inicializaci, protože metoda __new__() v rodičovské třídě je dostatečná a zavolá se automaticky.

6.3 UKÁZKA PRINCIPŮ OOP NA PŘÍKLADECH

Příklady principů objektově orientovaného programování jsou v elektronickém kurzu vysvětleny na čistě použitém kódu v Pythonu bez knihovny Kivy. Účelem příkladů je bezvýhradné pochopení principů, nechceme tedy kód zbytečně narušovat nepodstatnými prvky.

6.3.1 DĚDIČNOST – UKÁZKA

Pro ukázku dědičnosti je nutné vytvořit rodičovskou třídu a dva potomky. Rodičovská třída se jmenuje Zvíře a bude obsahovat inicializační metodu a metodu potrava s jedním parametrem (vyjma self). Úkolem metody bude vypsat, zda daný objekt má, nebo nemá rád zadaný druh potravy.

```
class Zvire:
    def __init__(self, jmeno):
        self.jmeno = jmeno

    def potrava(self, potrava):
            print("{} má ráda {}!".format(self.jmeno, potrava))
```

Potomkem této třídy bude třída Kočka, která bude mít pro ni speciální metodu zamňoukej.

```
class Kocka(Zvire):
    def zamnoukej(self):
        print("{}: Mňau!".format(self.jmeno))
```

Druhým potomkem bude třída Pes, která bude mít metodu potrava a zaštěkej. Součástí metody pro potravu je navíc volání metody potrava z rodičovské třídy pomocí klíčového slova super.

```
class Pes(Zvire):
    def zastekej(self):
        print("{}: Haf Haf!".format(self.jmeno))

    def potrava(self, potrava):
        print("{} nesmí {}!".format(self.jmeno, potrava))
        super().potrava(potrava)
```

Na závěr vytvoříme objekty typu Kočka a Pes a vyvoláme jejich metodu potrava.

```
micinka = Kocka("Micinka")
micinka.potrava("ryby")
azor = Pes("Azorek")
azor.potrava("maso")
```

Jak je patrné, vytvořený objekt je inicializován díky metodě init v rodičovské třídě. Pokud tedy je metoda v rodičovské třídě vhodně definována, může ji potomek používat. Také je nutné si uvědomit, že rodičovská třída by měla obsahovat jen to, co platí pro všechny její potomky. Pokud se podíváte na uvedenou třídu Pes, vidíte, že tato třída obsahuje stejnou metodu jako rodičovská třída. Je tedy možné si takovou metodu vždy upravit pro naše potřeby. Někdy se může stát, že se nám hodí původní metoda a chceme ji jen rozšířit. K tomu nám slouží klíčové slovíčko super, které zavolá metodu z rodičovské třídy. Ve výsledku nám program vypíše tři řádky, kde dva budou pro objekt třídy Pes. První bude pro kočku, která metodu potrava nemá, tak se automaticky použije metoda z rodičovské třídy. Pes bude vypsán dvakrát, protože ve třídě Pes pomocí metody potrava nejdříve voláme výpis metody "Pes nesmí maso", ale vzápětí pomocí slovíčka super voláme metodu potrava z rodičovské třídy, která způsobí další výpis.

6.3.2 POLYMORFIZMUS – UKÁZKA

Principem polymorfizmu je to, že se všemi svými potomky můžeme komunikovat pomocí stejného rozhraní jako s rodičovskou třídou. Rodičovská třída má metodu, kterou bude mít i každý její potomek, ale bude mít jiný obsah. Princip je vysvětlen na obdobném příkladu jako dědičnost.

```
class Zviratko:
    def __init__(self, jmeno):
        self.jmeno = jmeno
```

```
    def potrava(self, jidlo):
        pass


class Kotatko(Zviratko):
    def potrava(self, jidlo):
        print("{}: mám ráda {}".format(self.jmeno, jidlo))


class Stenatko(Zviratko):
    def potrava(self, jidlo):
        print("{}: nemám rád {}".format(self.jmeno, jidlo))


zviratka = [Kotatko('Micka'), Stenatko('Azorek')]

for Zviratko in zviratka:
Zviratko.potrava('ryby')
```

V uvedeném kódu vidíme, že rodičovská třída má metodu potrava, ale samotná metoda není definována. Pokud tedy zavoláme metodu potrava z rodičovské třídy, tak se nic nestane. Obecná metoda se nám pro zvířata nehodí, pokud tedy budeme vytvářet potomka, tuto metodu již předně definujeme. Pes i kočka a libovolná další podtřída je zvíře. Díky tomu si můžeme udělat seznam a v podstatě už nám nezáleží na tom, o které zvíře se jedná. Díky tomu pracujeme s konkrétními zvířaty stejně jako s rodičovskou třídou. Jestliže program někde očekává zvíře, můžeme použít jak psa, tak kočku, protože oboje je zvíře.

6.3.3 ZAPOUZDŘENÍ – UKÁZKA

Python nemá soukromé klíčové slovo a spoléhá se na zapouzdření. Proměnná, která nemá být přímo přístupná, má být označena podtržítkem na začátku. Pro vyzkoušení si tedy vytvoříme třídu osoba, která bude mít tři proměnné.

```
class Osoba():
    def __init__(self):
        self.jmeno = 'Karel'
        self._bydliste = 'Plzen'
        self.__datum_narozeni = '1.1.1970'


karel = Osoba()
print(karel.jmeno)
print(karel._bydliste)
print(karel.__datum_narozeni)
```

Každá proměnná obsahuje různý počet podtržítek (0-2). Pokud máme objekt třídy Osoba, je možné zkusit přistupovat k daným proměnným. Přístup ke jménu a bydlišti je funkční, ale přístup k datu narození již nikoli. Zobrazí se chyba, že takový atribut v dané třídě neexistuje. Aby došlo k upřesnění, jedno podtržítko také značí soukromou proměnnou,

ale nic nám nebrání s ní pracovat. Samozřejmě ale dochází k porušení konvence. Dvojité podtržítko už proměnnou chrání lépe, ale stále je možnost se k proměnné dostat.

V jiné třídě Osoby je ukázáno, jak bychom měli pracovat se soukromými atributy podle konvence. Využívá se k tomu technika getrů a setrů, která je obdobná jako v jazyce Java.

```
class Osoby():
    def __init__(self):
        self.__datum_narozeni = '1.1.1970'

    def getDatumNarozeni(self):
        print(self.__datum_narozeni)

    def setDatumNarozeni(self, datum):
        self.__datum_narozeni = datum

pavel = Osoby()
pavel.getDatumNarozeni()
pavel.setDatumNarozeni('5.1.1970')
pavel.getDatumNarozeni()
```

Výše tedy vidíme, jak by měla konstrukce pro získání nebo změnu atributu vypadat. Samozřejmostí je kontrola relevantnosti údajů v metodě setDatumNarozeni. Toto není rozhodně jediný způsob, jak zapouzdření řešit, jiným způsobem pro řízení přístupu jsou dekorátory.

6.4 POKROČILÉ APLIKACE

Některé aplikace v této sekci se zabývají přímo objektově orientovaným programováním a některé jsou spíše orientované na jejich praktické využití. Tyto příklady by studentům měly ukázat, že jim jejich znalosti, které již získali, nebo v tomto bloku ještě získají, umožňují vytvářet velké množství zajímavých aplikací.

Například aplikace zaměřená na objektově orientované programování se jmenuje peněženka. Při vytváření této aplikace si samostatně vyzkouší implementaci metod get a set. Během úkolu je student nucen vytvořit vlastní třídu Ucet, která má veřejnou proměnnou jméno účtu a soukromou hodnotu obnos, který se na účtu nachází. Je nutné také v aplikaci ošetřit neplatný vstup pomocí výjimky. Složitějším prvkem této aplikace je vymyslet, jak při každém přidání/odebrání peněz z účtu vypisovat aktuální stav na účtu. Za tímto účelem musíme obnos definovat v hlavní střídě jako prvek StringProperty z knihovny kivy.properties. Dále definujeme metodu, která reaguje na změnu obnosu.

Z pohledu aplikace zásadně zaměřené na praktické využití je aplikace lidé, která v sobě skrývá mnoho z předchozích zkušeností. Aplikace nabízí přepínání mezi více obrazovkami,

dále také vyskakovací okno s informacemi. Nové a rozšiřující je v tomto použití SQLite databáze. Cílem úkolu je vytvořit aplikaci, která bude komunikovat s databází, informace se do databáze budou zapisovat, aktualizovat a také se z ní mazat. V aplikaci bude navíc možnost vypsat celý obsah databáze.

6.5 SQLITE DATABÁZE

SQLite databáze se vyznačuje tím, že pracuje jako samostatný proces uvnitř aplikace a není nutné používat princip klient/server. Předností této databáze je, že funguje jako proces, který běží jen v případě potřeby.

Databáze má podobu velmi malé knihovny napsané v jazyce C. Zásadní výhodou této databáze je, že nepotřebujeme připojení k internetu[24]. Jedná se tak o databázová řešení, která jsou vhodná pro výuku. Za další výhodu lze rozhodně považovat to, že lze tuto databázi používat na operačních systémech Windows, Linux i MacOs.

Tento typ databáze patří mezi relační. To znamená, že je databáze založena na tabulkách. Každá tabulka obsahuje položky jednoho typu. Typem může být například námi navržená třída. Databáze většinou neobsahuje pouze jednu tabulku, ale více, které mají mezi sebou různé vazby. Pokud například budeme mít tabulku autoři knih, tak k tomuto záznamu bude patřit vícero záznamů v tabulce knihy. Záznamy jsou vždy ukládané do řádky tabulky a jednotlivé sloupce jsou různé atributy, které záznamy mají. SQLite databáze není typovaná, a tudíž nemusíme uvádět, jakého datového typu jednotlivé sloupce jsou. Každý záznam v tabulce by měl být označen unikátním identifikátorem. Většinou se tento identifikátor označuje id, to musí být v databázi jedinečné.

Samotná databáze ale není jen úložiště dat. Jedná se o odladěný a sofistikovaný systém. I když to zvenku není vidět a ani to nepoznáme, tento systém za nás řeší mnoho problémů. Pro komunikaci s databází používáme SQL jazyk, který je v podstatě tvořen snadno pochopitelnými větami. Vlastnosti databáze jsou zahrnuty ve zkratce ACID.

- Atomicity – pokud je v transakci více dotazů a jeden z nich selže, tak se databáze vrátí do stavu před tímto dotazem
- Consistency – databáze je vždy po dokončení dotazu konzistentní, opačný stav nemůže nikdy nastat
- Isolation – dotazy jsou zpracovány jako ve frontě

[24]About SQLite. *SQLite* [online]. [cit. 2019-04-08]. Dostupné z: https://www.sqlite.org/about.html

- Durability – každý změna v databázi je okamžitě uložena[25]

Pro práci a vytváření databáze bez znalosti příkazů nám slouží program DB Browser for SQLite. Jedná se o velmi oblíbenou aplikaci pro prohlížení, úpravu i vytváření databází v grafickém prostředí. Aplikace je ke stažení na sqlitebrowser.org/.

6.6 KONKRÉTNÍ ÚKOL POKROČILÉ APLIKACE – SQLITE DATABÁZE

Tato aplikace již dostane velmi reálné obrysy běžné praxe. V dnešní době se většina informací ukládá do databáze, a tak si to vyzkoušíme i my. Použijeme k tomu databázi SQLite, která se vyznačuje tím, že není typu klient-server, ale jedná se o knihovnu, která nám vytváří databázi na lokálním disku. Další výhodou této databáze je nezávislost na platformě.

Úkol: Vytvoříme aplikaci, která bude mít menu pro zvolení, zda chceme data přidat, zobrazit, upravit nebo smazat. Po kliknutí na položku se zobrazí jiné okno, kde budou potřebné prvky. Například pro přidání nám bude stačit pro ukázku jedno vstupní pole pro jméno. Dále zde budou tlačítka pro přidání jména do databáze a tlačítko zpět. Po přidání se zobrazí informační okénko o tom, zda se přidání zdařilo či ne. V tomto úkolu se počítá s tím, že bude databáze ručně vytvořena. Samozřejmě je možné databázi a její tabulky vytvořit pomocí příkazů v Pythonu. Jak na to, naleznete zde: https://likegeeks.com/python-sqlite3-tutorial/ Doporučuji, si rozhodně tento zdroj projít.

Novými prvky tohoto úkolu jsou práce s databází a vyskakovací okénko. Databázi musíme nejdříve k aplikaci připojit a poté s ní můžeme pracovat. V aplikaci musíme importovat knihovnu sqlite3. Pro připojení použijeme následující příkazy:

```
con = sqlite3.connect('db/lide.db')
cursor = con.cursor()
```

Pro práci s databází se využívá standardní dotazovací SQL jazyk a metoda pro přidání může vypadat takto:

```
def data_pridat(jmeno, idu):
cursor.execute("insert into lide values(?,?)", (jmeno, idu))
con.commit()
```

Nejdříve jsme tedy vytvořili příkaz, který se má provést, a následně jsme ho předložili databázi k provedení.

[25]ACID Properties in DBMS. *GeeksForGeeks* [online]. [cit. 2019-04-08]. Dostupné z: https://www.geeksforgeeks.org/acid-properties-in-dbms/

Příkaz pro aktualizování dat a smazání dat následovně:

```
Update lide set jmeno=? where id=?
Delete from lide where id=?
```

Poslední relevantní SQL dotaz je pro získání všech dat z tabulky:

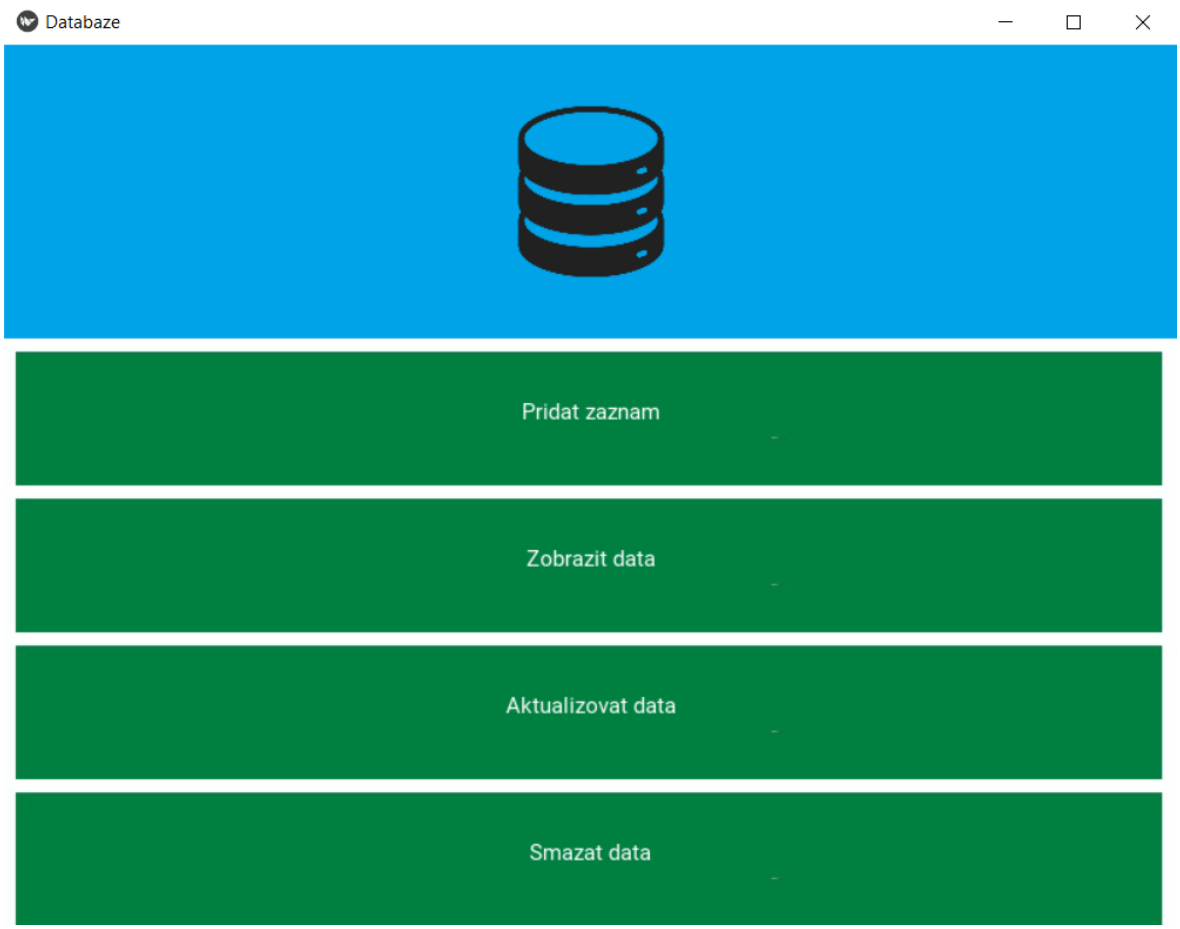
```
Select * From lide
```

Pro zobrazení okénka s informací, zda se operace provedla správně, použijeme prvek ModalView. Vytvoříme si vlastní návrh prvku a následně vytvoříme jeho instanci.

```
class Modal(ModalView):
labeltext = StringProperty()

def __init__(self, labeltext, **kwargs):
super(Modal, self).__init__(**kwargs)
self.labeltext = labeltext
```

Posledním krokem je nastavení okénka v souboru KV, kde je nejdůležitější pro zničení okénka použít metodu dismiss().



Obrázek 11: Ukázka možného výsledného řešení (zdroj: vlastní)

6.7 POPIS VYTVÁŘENÍ ZDROJOVÉHO KÓDU APLIKACE

V této aplikaci již vytvoříme pokročilejší uživatelské rozhraní, kam jak na pozadí aplikace, tak i tlačítek vložíme obrázek. Dále musíme vytvořit několik obrazovek a vyskakovací okénko. Hlavní okénko aplikace bude menu. Podle vybrané části aplikace se zobrazí následující obrazovka. Menu bude obsahovat tlačítka pro přesun mezi obrazovkami: přidat, zobrazit, aktualizovat a smazat. Pokud se tedy pustíme do vytváření uživatelského rozhraní, nastavíme nejdříve prvku Button výchozí hodnoty. První výchozí hodnota bude pro roztažení tlačítka na 100 % a nastavení pozadí tlačítka, což je v našem případě obrázek, kde v bílém rámečku je zelené pozadí. Následně si definujeme vzhled vyskakovacího okénka, které bude vždy stejné a měnit se bude jen uvedený text. Za tímto účelem využijeme vlastní návrh třídy, která bude založena na třídě kivy.uix.modalview. Vyskakovacímu okénku nastavíme následující atributy. Velikost bude 50 % na obou osách původního okna. Pro zobrazení vlastního obrázku na pozadí tohoto okénka musíme vložit do okénka na pozadí plátno, které se bude rozkládat přes celé okénko, a vložit na něj obdélník, v němž se zobrazí náš obrázek pozadí. Nyní přidáme na okénko BoxLayout, který bude orientován vertikálně a bude mít prvky Label a Button. Label bude zobrazovat text, který bude nastaven v hlavní třídě, a opět přidáme na jeho pozadí obrázek pozadí jako v případě tohoto okénka. Tlačítku nastavíme nižší výšku, aby nebylo zbytečně velké, a bude sloužit k zavření okénka, tudíž je při stisknutí tlačítka nutné vyvolat funkci dismiss.

Vše potřebné již máme připraveno a můžeme se pustit do vytváření všech oken aplikace. Třída pro hlavní okno musí být typu ScreenManager a ostatní typu Screen, všem opět musíme přidat náš obrázek na pozadí. Hlavní okno bude obsahovat BoxLayout orientovaný vertikálně. Prvním prvkem bude obrázek představující databázi a pod ním budou tlačítka, která při stisknutí vyvolají a zobrazí jinou obrazovku. Vzhled obrazovky pro přidání do databáze bude obsahovat šablonu rozložení, popisek nad vstupním políčkem a tlačítko pro návrat na hlavní obrazovku a pro uložení záznamu do tabulky. Tlačítko vyvolá funkci uložit, kterou budeme později implementovat pomocí Python kódu. Obrazovka pro zobrazování bude obsahovat ListView pro zobrazení všech záznamů z databáze a opět tlačítko pro návrat na hlavní obrazovku a tlačítko, které vyvolá funkci pro zobrazení veškerých záznamů z databáze. Obrazovka pro aktualizaci dat bude obsahovat dvě vstupní pole, kde v jednom z nich uvedeme id záznamu a v druhém poli

bude prostor pro změnu záznamu. Ve spodní části se budou nacházet tlačítka pro návrat a pro vyvolání funkce k aktualizaci záznamu. Poslední obrazovka bude obsahovat vstupní pole pro zadání id záznamu, který má být z databáze odstraněn, a opět tlačítka pro návrat na hlavní obrazovku a pro vyvolání funkce ke smazání záznamu.

K vytvořenému grafickému rozhraní vytvoříme funkční kód v Pythonu. Nejdříve si vytvoříme třídu, která v grafickém rozhraní obsahuje všechny obrazovky aplikace, tudíž tato třída musí být potomkem třídy ScreenManager. Ostatní třídy představující jednotlivé obrazovky musí být potomky třídy Screen. Dosud uvedené třídy nepotřebují žádný speciální obsah a stačí je tedy definovat slovíčkem pass. Je nutné importovat potřebné třídy, zároveň můžeme vytvořit i import pro knihovnu sqlite3 a ModalView z knihovny kivy.uix. Vytvoříme tedy třídu založenou na třídě ModalView a definujeme ji. Jelikož chceme, aby vyskakovací okno zobrazovalo pokaždé jiný text, vytvoříme si proměnnou, která je typu StringProperty. To způsobuje, že se obsah této proměnné automaticky aktualizuje, pokud se tento atribut objektu změní. Následně redefinujeme magickou metodu __init__ tak, že přidáme parametry v podobě našeho textu a **kwargs. Kwargs nám umožňuje předat klíčovou hodnotu a přidat ji do slovníku objektu pod vlastním názvem. Přímo v metodě zavoláme přes slovíčko super inicializační metodu z třídy ModalView s parametry kwargs a na další řádce definujeme vlastní proměnnou potřebného textu. Je nutné definovat proměnnou, která bude vytvářet připojení k naší databázi, a proměnnou, která bude na danou databázi ukazovat. Dalším krokem je definování metod pro práci s databází. Metoda pro přidání má parametry jméno a id uživatele. Přes ukazatel na databázi a příkaz execute vyvoláme SQL funkci pro přidání uživatele do tabulky. Příkaz by mohl vypadat tímto způsobem: "'insert into lide values(?,?)', (jmeno, id))". Laicky řečeno kód říká: vlož do tabulky lidé záznam s těmito hodnotami. V tuto chvíli je databáze pozměněna, ale změnu vidíme lokálně jen my. Pro potvrzení změny je nutné přes proměnnou databáze vyvolat příkaz commit, který změní databázi i pro ostatní uživatele. Metody pro aktualizaci a mazání mají stejnou konstrukci, ale odlišný SQL příkaz. Klíčová slovo pro aktualizaci jsou update a set, pro smazání delete a from. Posledním krokem je vytvoření hlavní třídy, která je typu App u knihovny Kivy. Jako vždy je nutné definovat metodu build, která bude vracet třídu, která se stará o celkovou grafickou stránku aplikace. V tuto chvíli definujeme metody pro přidání,

zobrazení, aktualizaci a smazání obsahu z databáze. Metoda pro zobrazení dat bude nejdříve volat proměnnou list_stu, kterou definujeme v grafické části a jejímž úkolem je vyčistit její obsah. Pokud bychom tak neučinili, data by se nám při zobrazování opakovala a nepoznali bychom aktuální obsah databáze. Přes ukazatel na databázi zavoláme příkaz pro získání veškerých dat v databázi, která pomocí cyklu přidáváme do proměnné list_stu. V tomto případě se může stát, že je databáze prázdná. Pokud se přesuneme k dalším metodám, zde je již nutné pomocí výjimky hlídat, zda nedošlo k nějaké chybě. Zavoláme dané funkce a následně vytvoříme instanci vyskakovacího okénka, kterou je nutné ještě zobrazit pomocí příkazu open. Obsah okénka bude ovlivněn tím, jestli byla instance vyvolána v části try nebo except. Poslední metoda je nutná pro přepínání mezi obrazovkami aplikace. To je například možné vyřešit pomocí proměnné. Pokud tedy v grafickém prostředí stiskneme tlačítko, tak se zavolá naše metoda se zadaným parametrem. Parametr můžeme postupně porovnávat pomocí podmínek, a pokud nalezneme shodu, vyvoláme zobrazení příslušné obrazovky. Výsledek k nahlédnutí v příloze.

7 ZÁVĚR

Prudký rozvoj společnosti a mobilních zařízení dnes značně utlumuje čas strávený u klasických strojů jako počítač a notebook. Dnes je již běžné, že značná část společnosti využívá chytré telefony a tablety. Pokud chceme vytvořit kvalitní a oblíbené prostředí, je vhodné připravit jednu aplikaci pro co nejvíce platforem, což nám Python a knihovny Kivy umožňují v minimálním čase. Aplikaci stačí vytvořit jednou a bez úprav ji vygenerovat pro různá prostředí.

V této práci jsem se zaměřil na vývoj samotných Android aplikací, uvádím nutné vstupní kroky a nabízím čtenářům více možností, aby si samostatně mohli vybrat vývojový nástroj. Součástí práce je také zaměření na objektově orientované programování, které je v dnešní době nejrozšířenějším přístupem k tvorbě aplikací.

V praktické části jsem vytvořil elektronický výukový kurz, který má za cíl pokročilé středoškoláky a vysokoškoláky uvést do problematiky vývoje Android aplikací pomocí Pythonu. Jako vstupní zkušenost se předpokládá práce s programovacím jazykem Python. Vytvářené aplikace jsem se snažil dělat atraktivní, aby byl rozvoj náročnosti aplikací pozvolný. Věřím, že student, který zvládne absolvovat kurz, bude schopen sám implementovat další složité aplikace s pomocí technik, které se v průběhu kurzu naučil. Součástí kurzu je i krátké seznámení s databázemi a jazykem SQL.

Úkoly, které jsem navrhl, jsem také sám zpracoval. V kurzu jsem umožnil studentům řešení k nahlédnutí. Ovšem v programování vždy existuje mnoho cest k řešení. Dále jsem studentům navrhl, jak by mohli vyřešit nastalý problém v aplikaci. Člověk v tomto oboru by měl být kreativní, a proto jsem stanovil, že závěrečná práce bude založena na výběru samotného studenta. Student, případně dvojice studentů vymyslí cíl aplikace a zkusí ji navrhnout. Jedním z cílů bude vedoucímu kurzu dostatečně objasnit účel a formu aplikace. Pokud se studenti nedokážou s nějakou chybou vypořádat, bude jim vedoucí kurzu sloužit jako pomoc, která může navrhnout nové řešení, nebo přivede studenty na novou myšlenku.

RESUMÉ

Práce je zaměřena na vývoj Android aplikací v Pythonu. Praktická část práce spočívala ve vytvoření elektronického výukového kurzu na toto téma. Cílem práce je seznámit čtenáře se základními informacemi o programovacím jazyku Python 3, programovými nástroji na vývoj Android aplikací pomocí Pythonu a knihovny Kivy. V druhé kapitole jsou popsány vybrané programové nástroje, a to Kivy Designer, Python IDLE a Visual Studio Code. Pro výukový kurz byl vybrán vývojový nástroj PyCharm od velmi známého vydavatele vývojových prostředí, společnosti JetBrains. Pátá kapitola práce je zaměřena na základy vývoje Android aplikací pomocí Kivy, včetně návrhu jednodušších aplikací. Závěr této kapitoly obsahuje podrobný postup, jak z aplikace vytvořit APK balíček pro mobilní zařízení s Androidem. Poslední kapitola práce je zaměřena na principy objektově orientovaného programování, které jsou zde popsány teoreticky, ukázány prakticky a následně zapojeny do praktických úkolů. Pro zvýšení atraktivity kurzu, práce a složitějších aplikací byl do práce přidán základní článek zaměřený na SQLite databázi. Po jeho prostudování studenti kurzu přistoupí k aplikaci, kde si práci se SQLite databází vyzkouší.

SUMMARY

This Thesis deals with development of Android applications in Python. The practical part of the Thesis was to create an electronic training course on this topic. The aim is to inform the readers on basic programming in Python 3 language and some programming tools for development of Android appliactions with help of Python and Kivy library.

In the second chapter some specific programming tools are described, such as Kivy Designer, Python IDLE and Visual Studio Code. For the educational course a development tool called PyCharm by a well known development environments publisher JetBrains was selected. The fifth chapter is focused on basic skills of development of Android applications per Kivy including programming of simple applications. The final part of this chapter shows a detailed procedure how to create an APK package for a mobile device run on Android based on an application.

The Last chapter deals with object oriented programming principles which are described in theory there, shown practically and then included in hands-on tasks. To increase attractiveness of the course, the Thesis as well as more complicated applications a basic article about SQLite database was added. After reading this the participants of the course will be invited to use an application to work with SQLite database.

SEZNAM LITERATURY

SUMMERFIELD, Mark. *Python 3*: výukový kurz. Brno: Computer Press, 2013, s. 13-13. ISBN 978-80-251

LACKO, Ľuboslav. *Mistrovství – Android*. Přeložil Martin HERODEK. Brno: Computer Press, 2017. Mistrovství. ISBN 978-80-251-4875-4.

PHILLIPS, Dusty. *Creating apps in Kivy*. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1-491-94667-1.

VASILKOV, Mark. *Kivy Blueprints*. Birmingham: Packt Publishing Ltd., 2015. ISBN 978-1-78398-784-9-2737-7.

Python Documentation by Version. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.python.org/doc/versions/

Create a package for Android. *Kivy.org* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/guide/packaging-android.html

Build a Python Web Server with Flask. *Raspberry Pi* [online]. [cit. 2019-04-08]. Dostupné z: https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/2

CARBONNELLE, Pierre. PYPL PopularitY of Programming Language. *PYPL* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://pypl.github.io/PYPL.html

LONG, Moe. 6 Easiest Programming Languages to Learn for Beginners. *MakeUseOf* [online]. 2016, 29.12.2016 [cit. 2019-04-08]. Dostupné z: https://www.makeuseof.com/tag/easiest-programming-languages-beginners/

CLEARY, Annabel. Top 5 Programming Languages for Beginners. *CoderDojo* [online]. 2015, 20.3.2015 [cit. 2019-04-08]. Dostupné z: https://coderdojo.com/2015/03/20/top-5-programming-languages-for-beginners/

TIOBE Index. *TIOBE* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.tiobe.com/tiobe-index/

Python.cz [online]. [cit. 2019-04-08]. Dostupné z: https://python.cz/

Kivy Designer. *GitHub* [online]. 2018 [cit. 2019-04-08]. Dostupné z: https://github.com/kivy/kivy-designer

Python Documentation by Version. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://www.python.org/doc/versions/

IDLE. *Python Software Foundation* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://docs.python.org/3/library/idle.html

March 2019 (version 1.33). *Visual Studio Code* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://code.visualstudio.com/updates/v1_33

About Company. *JetBrains* [online]. [cit. 2019-04-08]. Dostupné z: https://www.jetbrains.com/company/

CARBONNELLE, Pierre. Top IDE index. *PYPL* [online]. 2019 [cit. 2019-04-08]. Dostupné z: https://pypl.github.io/IDE.html

About Customers & Awards. *JetBrains* [online]. [cit. 2019-04-08]. Dostupné z: https://www.jetbrains.com/company/customers/

HOFMAN, Martin. *Vývoj Android aplikací v prostředí IntelliJ*. Plzeň, 2017. Bakalářská práce. Západočeská univerzita v Plzni.

Kivy: Cross-platform Python Framework. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/#home

Layouts. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/gettingstarted/layouts.html

HOFMAN, Martin. *Vývoj Android aplikací v prostředí IntelliJ*. Plzeň, 2017. Bakalářská práce. Západočeská univerzita v Plzni.

Kivy Language. *Kivy* [online]. [cit. 2019-04-08]. Dostupné z: https://kivy.org/doc/stable/api-kivy.lang.html

Lekce 1 - Úvod do objektově orientovaného programování v Pythonu. ITnetwork.cz [online]. [cit. 2019-04-08]. Dostupné z: https://www.itnetwork.cz/python/oop/python-tutorial-uvod-do-objektove-orientovaneho-programovani

About SQLite. *SQLite* [online]. [cit. 2019-04-08]. Dostupné z: https://www.sqlite.org/about.html

ACID Properties in DBMS. *GeeksForGeeks* [online]. [cit. 2019-04-08]. Dostupné z: https://www.geeksforgeeks.org/acid-properties-in-dbms/

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ A DIAGRAMŮ

PŘÍLOHY

Přílohy 1 – DVD

Příloha 2 – Kresleni.py

Příloha 3 – PoslaniMailu.py

Příloha 4 – PoslaniMailu.kv

Příloha 5 – SQLiteDB.py

Příloha 6 – Databaze.kv

Příloha 7 – lide.db

Příloha 1 – DVD

Na DVD se nachází:

- Adresář s projekty vytvořenými v rámci praktické části
- Vlastní text diplomové práce ve formátu docx a pdf

Příloha 2 – Kresleni.py

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.widget import Widget
from kivy.graphics import Color, Ellipse, Line


class MyPaintWidget(Widget):

    def on_touch_down(self, touch):
        with self.canvas:
            Color(0, 1, 0)
            d = 35
            Ellipse(pos=(touch.x - d / 2, touch.y - d / 2), size=(d, d))
            touch.ud['line'] = Line(points=(touch.x, touch.y), width=3)

    def on_touch_move(self, touch):
        touch.ud['line'].points += [touch.x, touch.y]


class Apka2(App):

    def clear_canvas(self, obj):
        self.platno.canvas.clear()

    def build(self):
        self.platno = MyPaintWidget()
        layout = GridLayout(rows=2)
        btn1 = Button(text='Smazat', size_hint_y=None, height=50)
        btn1.bind(on_release=self.clear_canvas)
        layout.add_widget(btn1)
        layout.add_widget(self.platno)

        return layout


if __name__ == "__main__":
    Apka2().run()
```

Příloha 3 – PoslaniMailu.py

```
import smtplib
import ssl

from kivy.app import App
from kivy.lang import Builder
from kivy.uix.screenmanager import Screen


class MainScreen(Screen):
    def poslat_mail(self, mujmail, mojeheslo, komu, predmet, text):
        port = 587
        smtp_server = "smtp.gmail.com"
        sender_email = mujmail
        password = mojeheslo
        receiver_email = komu

        message = "Subject: {}\n\n{}".format(predmet, text)

        context = ssl.create_default_context()
        with smtplib.SMTP(smtp_server, port) as server:
            server.starttls(context=context)
            server.login(sender_email, password)
            server.sendmail(sender_email, receiver_email, message)


presentation = Builder.load_file("PoslaniMailu.kv")


class MainApp(App):
    def build(self):
        return presentation


if __name__ == "__main__":
    MainApp().run()
```

Příloha 4 – PoslaniMailu.kv

```
MainScreen:
<MW@Widget>:
    id: separator
    size_hint_y: None
    height: 6
    canvas:
        Color:
            rgb: 1., 0., 0.
        Rectangle:
            pos: 0, separator.center_y
            size: separator.width, 2
<MainScreen>:
    name: 'main'
    BoxLayout:
        orientation: 'horizontal'
        BoxLayout:
            orientation: 'vertical'
            Label:
                text: 'Muj mail:'
            MW:
            Label:
                text: 'Moje heslo:'
            MW:
            Label:
                text: 'Prijemce:'
            MW
            Label:
                text: 'Predmet:'
            MW:
            Label:
                text: 'Text mailu:'
            MW:
            Label:
        BoxLayout:
            orientation: 'vertical'
            TextInput:
                id: mujmail
                text: ''
            TextInput:
                id: mojeheslo
                password: True
                text: ''
            TextInput:
                id: komu
                text: ''
            TextInput:
                id: predmet
                text: ''
            TextInput:
                id: textzpravy
                text: ''
            Button:
                text: 'Odeslat'
                on_release: root.poslat_mail(mujmail.text, mojeheslo.text,
             komu.text, predmet.text, textzpravy.text)
```

Příloha 5 – SQLiteDB.py

```
#-*-coding: utf-8 -*-"
import sqlite3
import random
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.modalview import ModalView
from kivy.properties import StringProperty
con = sqlite3.connect('db/lide.db')
cursor = con.cursor()


def data_pridat(jmeno, idu):
    cursor.execute("insert into lide values(?,?)", (jmeno, idu))
    con.commit()


def data_aktualizovat(nove_jmeno, id_uz):
    cursor.execute("Update lide set jmeno=? where id=?", (nove_jmeno, id_uz))
    con.commit()


def data_smazat(id_uz):
    cursor.execute("Delete from lide where id=?", (id_uz,))
    con.commit()


class Modal(ModalView):
    labeltext = StringProperty()

    def __init__(self, labeltext, **kwargs):
        super(Modal, self).__init__(**kwargs)
        self.labeltext = labeltext


class Main(ScreenManager):
    pass


class Pridat(Screen):
    pass


class Aktualizovat(Screen):
    pass


class Smazat(Screen):
    pass


class Zobrazit(Screen):
    pass


class Hlavni(Screen):
    pass


class Databaze(App):
    def build(self):
        return Main()

    def ulozit(self):
        try:
            jmeno = self.root.ids.ti_jmeno.text
```

```
            idu = random.randrange(999)
            data_pridat(jmeno, idu)
            m = Modal(labeltext="Uspesna registrace")
            m.open()
        except:
            m = Modal(labeltext="Neuspesna registrace")
            m.open()

    def zobraz_data(self):
        self.root.ids.list_stu.adapter.data.clear()
        oku = cursor.execute("Select * From lide")
        for i in oku.fetchall():
            self.root.ids.list_stu.adapter.data.append(str(i))

    def obrazovky(self, obrazovka):
        if obrazovka == "pridat":
            self.root.current = "pridat"
        elif obrazovka == "zobrazit":
            self.root.current = "zobrazit"
        elif obrazovka == "hlavni":
            self.root.current = "hlavni"
        elif obrazovka == "aktualizovat":
            self.root.current = "aktualizovat"
        elif obrazovka == "smazat":
            self.root.current = "smazat"

    def aktualizovat(self):
        try:
            id_uz = self.root.ids.ti_nove_jmeno.text
            nove_jmeno = self.root.ids.txtname.text
            data_aktualizovat(nove_jmeno, id_uz)
            mod = Modal(labeltext="Uspesna aktualizace")
            mod.open()
        except:
            mod = Modal(labeltext="Neuspesna aktualizace")
            mod.open()

    def smazat(self):
        try:
            id_uz = self.root.ids.ti_id_smazat.text
            data_smazat(id_uz)
            mod = Modal(labeltext="Uspesne smazani")
            mod.open()
        except:
            mod = Modal(labeltext="Neuspesne smazani")
            mod.open()


Databaze().run()
```

Příloha 6 – Databaze.kv

```
#-*-coding: utf-8 -*-"
#:import Label kivy.uix.label.Label
#:import SimpleListAdapter kivy.adapters.simplelistadapter.SimpleListAdapter
<Button>:
    size_hint:1,None
    background_normal:"btn1.jpeg"

<Modal>:
    size_hint:0.5,0.5
    canvas.before:
        Rectangle:
            pos:self.pos
            size:self.size
            source:'pozadi.jpeg'
    BoxLayout:
        orientation:"vertical"
        Label:
            text:root.labeltext
            canvas.before:
                Rectangle:
                    pos:self.pos
                    size:self.size
                    source:'pozadi.jpeg'
        Button:
            text:"Zavrit"
            on_press:root.dismiss()
            size_hint:1,0.3
<Main>:
    Hlavni:
        name:"hlavni"
        canvas.before:
            Rectangle:
                pos:self.pos
                size:self.size
                source:'pozadi.jpeg'
        BoxLayout:
            orientation:"vertical"
            Image:
                source:"db.png"
            Button:
                size_hint:1,None
                text:"Pridat zaznam"
                on_press:app.obrazovky("pridat")
            Button:
                size_hint:1,None
                text:"Zobrazit data"
                on_press:app.obrazovky("zobrazit")
            Button:
                size_hint:1,None
                text:"Aktualizovat data"
                on_press:app.obrazovky("aktualizovat")
            Button:
                size_hint:1,None
                text:"Smazat data"
                on_press:app.obrazovky("smazat")
    Pridat:
        name:"pridat"
        canvas.before:
            Rectangle:
                pos:self.pos
                size:self.size
                source:'pozadi.jpeg'
        BoxLayout:
            orientation:"vertical"
            spacing:10
```

```
            padding:5
            Label:
                text:"Jmeno"
            TextInput:
                size_hint:1,0.3
                multiline:False
                id:ti_jmeno
            BoxLayout:
                Button:
                    size_hint:0.5,None
                    text:"Ulozit"
                    on_press:app.ulozit()
                Button:
                    size_hint:0.5,None
                    text:"Zpet"
                    on_press:app.obrazovky("hlavni")
    Zobrazit:
        name:"zobrazit"
        canvas.before:
            Rectangle:
                pos:self.pos
                size:self.size
                source:'pozadi.jpeg'
        BoxLayout:
            orientation:"vertical"
            ListView:
                id:list_stu
                adapter:
                    SimpleListAdapter(data=[],
                    cls=Label)
            Button:
                text:"Zobrazit"
                on_press:app.zobraz_data()
                size_hint:1,None
            Button:
                size_hint:1,None
                text:"Zpet"
                on_press:app.obrazovky("hlavni")
    Aktualizovat:
        name:"aktualizovat"
        canvas.before:
            Rectangle:
                pos:self.pos
                size:self.size
                source:'pozadi.jpeg'
        BoxLayout:
            orientation:"vertical"
            spacing:10
            padding:5
            Label:
                text:"id dat"
            TextInput:
                size_hint:1,0.3
                multiline:False
                id:ti_nove_jmeno
            BoxLayout:
                orientation:"vertical"
                Label:
                    text:"Nova data"
                TextInput:
                    size_hint:1,0.3
                    multiline:False
                    id:txtname

            BoxLayout:
                Button:
                    size_hint:0.5,None
                    text:"Aktualizovat"
```

```
                on_press:app.aktualizovat()
            Button:
                size_hint:0.5,None
                text:"Zpet"
                on_press:app.obrazovky("hlavni")
Smazat:
    name:"smazat"
    canvas.before:
        Rectangle:
            pos:self.pos
            size:self.size
            source:'pozadi.jpeg'
    BoxLayout:
        orientation:"vertical"
        spacing:10
        padding:5
        Label:
            text:"id dat"
        TextInput:
            id:ti_id_smazat
            multiline:False
            size_hint:1,0.3
        BoxLayout:
            orientation:"vertical"
            Button:
                size_hint:1,None
                text:"Vymazat"
                on_press:app.smazat()
            Button:
                size_hint:1,None
                text:"Zpet"
                on_press:app.obrazovky("hlavni")
```

Příloha 7 – lide.db

```
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS "lide" (
             "jmeno"      TEXT,
             "id"   INTEGER PRIMARY KEY AUTOINCREMENT
);
INSERT INTO "lide" VALUES ('David',13);
INSERT INTO "lide" VALUES ('Lukas',422);
COMMIT;
```