ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA PEDAGOGICKÁ

KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

# Vývoj android aplikací v jazyce python

BAKALÁŘSKÁ PRÁCE

Roman Kulda

*Informatika se zaměřením na vzdělávání*

Vedoucí práce: PhDr. Denis Mainz, Ph.D.

Plzeň 2018

# Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této kvalifikační práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

.....

podpis

V Plzni dne 26.6.2018 Roman Kulda

## Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce PhDr. Denisovi Mainzovi, Ph.D za cenné rady, připomínky a metodické vedení práce.

# Zadání

# OBSAH

# SEZNAM ZKRATEK

ACW ..... Android Callable Wrappers

ADB ..... Android Debug Bridge

ADT ..... Android Development Tools

AOT ..... Ahead Of Time

API ..... Application programming interface

ART ..... Android Runtime

AVD ..... Android Virtual Devices

DVM ..... Dalvik Virtual Machine

HAL ..... Hardware Abstraction Layer

HTML ..... HyperText Markup Language

IDE ..... Integrated Development Environment

JAR ..... Java ARchive

JIT ..... Just In Time

MCW ..... Managed Callable Wrappers

MS ..... Microsoft

SD ..... Secure Digital (paměťová karta)

SDK ..... Software Devepment Kit

SE ..... Sony Ericsson

SSL ..... Secure Socket Layer

USB ..... Universal Serial Bus

WYSIWYG ..... What You See Is What You Get

XML ..... Extensible Makup Language

WAP ..... Wireless Application Protocol

# ÚVOD

Tvorba aplikací pro mobilní zařízení je v současné době jedním z nejvýznamnějších a nejvíce dynamicky se rozvíjejících odvětví programování. K tvorbě mnohdy rozsáhlých a propracovaných mobilních aplikací existuje celá řada vývojových prostředí, robustních nástrojů a programovacích jazyků. Ovšem ne všechny mobilní aplikace jsou rozsáhlé a komplikované, spotřebovávající velké objemy systémových prostředků a vyžadující pečlivé zacházení s omezeným množstvím paměťových a výpočetních kapacit. Současně ne všichni, a zejména ti začínající programátoři, jsou dostatečně zkušení a znalí tvořit komplikované aplikace prostřednictvím vyšších objektově orientovaných programovacích jazyků. V mnoha případech je dostačující a často také vhodnější použít jazyk a prostředí více přehledné, efektivní a programátorsky nenáročné s důrazem na univerzálnost a uživatelské rozhraní.

Předmětem této bakalářské práce je zejména zmapovat nabídku vývojových modulů pro programování multiplatformních aplikací v jazyce Python, který je právě jedním z představitelů vyšších objektově orientovaných jazyků používaných při práci s moduly vývojových nástrojů. Z následně provedeného výběru těch nejvýznamnějších variant konkrétně zaměřených modulů pak tyto varianty popsat a podrobněji posoudit z hlediska jejich možností v oblasti vývoje aplikací s grafickým uživatelským rozhraním pro mobilní operační systém Android. Srovnání a kritické vyhodnocení jednotlivých modulů bude provedeno také s ohledem na možnosti tvorby moderních aplikací s přívětivým uživatelským rozhraním. Stěžejním cílem této práce je pak zvolit z těchto možností vhodné řešení a na něm pak demonstrovat vytvoření jednoduchých ukázkových aplikací.

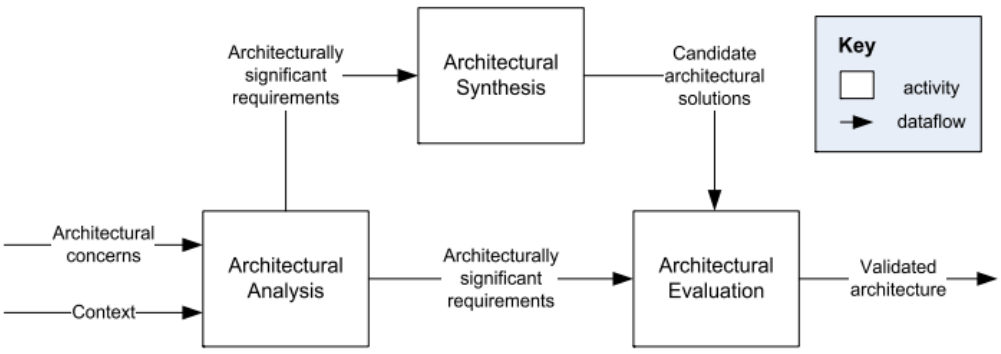
# 1 PROGRAMOVÁNÍ MOBILNÍCH APLIKACÍ

Mobilní aplikace v jednoduché formě se v mobilních zařízeních objevily se sklonkem 20. století. Již od počátku dodávaly mobilnímu telefonu vedle majoritní funkce komunikace další dimenze, konkrétně zpočátku v podobě kalendáře, kalkulátoru, budíku apod., k nimž později přibyly primitivní hry, což *„začala v roce 1998 Nokia, když nainstalovala do telefonu dnes již legendární hru Snake"*[1]. Aplikace v mobilních telefonech tehdy znamenaly využití technologie Java s aplikacemi s datovou strukturou JAR (Java ARchive). S poklesem cen mobilních přístrojů, vyšší kapacitou baterie a především nárůstem výpočetní kapacity mobilních zařízení došlo k velmi strmému nárůstu nabídky i složitosti aplikací. Významný krok představují implementace propracovanějších operačních systémů, z nichž mezi ty první patřil např. systém Palm OS a další, zejména pak Symbian 6 firmy Nokia[2]. Skutečný zlom pak přichází v letech 2007 a 2008, kdy se na trhu objevují první chytré mobilní telefony s dotykovými obrazovkami (iPhone, Samsung Galaxy) a k nim přidružené platformy iOS a Android.

## 1.1 Obecná architektura mobilních aplikací

Návrh softwarového produktu[3] je obecně proces hledání a v ideálním konečném výsledku nalezení, strukturovaného řešení, které splňuje veškeré technické a funkční požadavky na toto řešení. To vše při optimalizaci všech kvalitativních atributů, mezi něž patří běh aplikace a její a výkon, úroveň zabezpečení, možnosti údržby, optimalizace a dalšího vývoje či aktualizace. Dle jedné z definic architektury softwaru[4] je tato *„struktura komponent programu nebo systému, jejich vzájemné vazby, principy a předpisy určující jejich návrh a vývoj v průběhu času."* (pozn.: volně přeloženo autorem). Architekturu softwaru či výpočetního systému lze tedy chápat jako[5] uspořádání systému skládajícího se z prvků software, vlastností těchto prvků (služby, obsluha chyb, využití sdílených zdrojů, apod.) a vztahů mezi nimi.

[1] History of Nokia part 2: Snake.
[2] Symbian: příběh nejrozšířenějšího OS
[3] .NET application architecture guide.
[4] CLEMENTS, P. C. Coming Attractions in Software Architecture.
[5] BASS, L., CLEMENTS, P., KAZMAN, R. Software Architecture in Practice.




*Obr. 1: Proces tvorby softwarové architektury (zdroj: HOFMEISTER, C. & col.[6]).*

Kromě klasického algoritmu procesu tvorby architektury softwaru (viz obr. 1) stojí za zmínku i dva alternativní přístupy[7]: První z nich reprezentuje přístup hned od počátku návrhu přemýšlet o úroveň výš nad řešeným problémem, tedy volit obecnější výsledné řešení. Výhodou může být vyšší pravděpodobnost vytvoření skutečně funkčního řešení, které daný požadavek splní, nicméně je třeba ovšem zmínit riziko, že výsledkem může snadno být předimenzované řešení, které v konečném součtu nemusí být rentabilní a mnohdy také svou redundancí ne zcela uživatelsky přívětivé. Druhým možným přístupem je zcela opačný pohled, a sice od počátku se v myšlenkové rovině pohybovat o úroveň níž a řešit postupně dílčí problémy s představou, že spojením dílčích řešení dojde k vyřešení problému komplexního. Tento přístup je výhodný zejména v případech, kdy k řešené oblasti již existují komponenty, které jsou dostupné ke znovupoužití. Je tu však reálné riziko, že mozaika nebude jako celek fungovat zcela korektně, v krajním případě vůbec.

## 1.2 Výběr platforem pro návrh

Pro programování mobilních aplikací je možné zvolit z široké plejády systémů. Stěžejní otázkou k zodpovězení pro zúžení výběru je především cílový operační systém mobilního zařízení. V tomto případě je volba zřejmá, a sice operační systém Android. Vzhledem k tomu, že se jedná o s přehledem nejpoužívanější operační systém (jak ukazuje graf v kapitole 3 věnované operačnímu systému Android), existuje pro něj opět velmi pestrá nabídka platforem. Vzhledem k charakteru operačního systému je první logickou volbou nativní jazyk Java, potažmo jazyk C a z něj přímo odvozené verze (viz kap. 3). Ovšem tyto jazyky se vyznačují sice velkým výkonem a značnou silou pro řešení i těch nejnáročnějších aplikací, nicméně současně jsou poměrně náročné na schopnosti a zkušenosti programátora. Pokud by se tedy jednalo o programování jednodušší aplikace, která nebude využívat velký výpočetní

[6] HOFMEISTER, C. & col. A general model of software architecture design derived from five industrial approaches.
[7] TAYLOR, R., MEDVIDOVIČ, N., DASHOFY, E. Software Architecture: Foundations, Theory and Practice.

výkon či rozsáhlé komunikace s databázemi apod., velmi zajímavou a moderní volbou zejména pro méně zkušené programátory s potřebou vytvořit jednoduchou aplikaci může být jazyk Python. Tento jazyk sice není primárně určený pro vývoj mobilních aplikací, nicméně díky trvale rostoucí tendenci používání chytrých telefonů a tabletů jako osobní výpočetní jednotky, zejména pak na platformě Android, došel rozvoj i tohoto jazyka do fáze, kdy je díky celé řadě prostředí použitelný i pro tento účel. Jedním z prostředí, které je založeno na myšlence multiplatformity a může tak být použito k tvorbě aplikací pro v zásadě kterýkoli operační systém, je prostředí Kivy. Prostředí vykazuje nejen zmíněnou širokou podporu mnoha platforem, ale také jako užitečnou zajímavost v podobě proprietálního jazyka KV, který již tak přehlednému jazyku Python dodává ještě více na přehlednosti a strukturovanosti. Výběr systémů pro demonstraci tvorby mobilní aplikace představuje kombinace Python 3 a prostředí Kivy. Nutno ještě upřesnit, že jazyk Python existuje ve dvou verzích, a sice ve starší verzi 2. x a v novější, současné verzi 3.x. Problematice rozdílnosti obou jazyků se věnuje kap. 2.2.

# 2 PROGRAMOVACÍ JAZYK PYTHON

Jak již bylo zmíněno, jako základní programovací jazyk pro tvorbu mobilní aplikace byl zvolen Python.

## 2.1 Vybraná charakteristika a základní vlastnosti jazyka

Python představuje vysokoúrovňový programovací jazyk, který bývá často zařazován mezi skriptovací programovací jazyky, nicméně jde jen o část možností použití jazyka a kategorii skriptovacích jazyků Python značně přesahuje. Zásadní je fakt, že se jedná o hybridní, neboli také o víceparadigmatický, programovací jazyk, což vyjadřuje, že Python připouští procedurální programování, nebo také objektově orientovaný přístup (OOP, Object Orientet Programming) a zčásti také funkcionální programování, a to naprosto libovolně dle vůle a dispozic programátora, či dle vhodnosti s ohledem na řešený problém. Díky tomu disponuje Python vynikající variabilitou a velkými vyjadřovacími schopnostmi, díky čemuž je zdrojový kód jazyka velmi dobře čitelný, přehledný a efektivní, potažmo také krátký. Python je z pohledu způsobu překladu kódu dynamický interpretovaný jazyk a skýtá tedy dynamickou kontrolu datových typů. Python se snadno vkládá aplikací vytvořených v jiném jazyce (tzv. embedding), kde pak slouží, jak již bylo zmíněno, jako skriptovací jazyk. Toto bývá velmi často využíváno, což zavdává příčinu pro jeho zařazování mezi jazyky skriptovací. Naopak jiné aplikace či knihovny mohou zahrnovat rozhraní dovolující jejich použití jako modulu v prostředí Python.

S ohledem na výše uvedené je jednou ze signifikantních vlastností jazyka Python jeho efektivita a produktivnost z hlediska rychlosti psaní programů, a to počínaje jednoduchými programy s krátkým zdrojovým kódem, kde se zmíněná efektivita projevuje především stručností zápisu, až po rozsáhlé a komplexní aplikace, kde se efektivita Pythonu projevuje ještě zřetelněji. S vysokou produktivností souvisí dostupnost a snadná použitelnost široké škály knihovních modulů, umožňujících snadné řešení úloh z mnoha různých oblastí. K dalším příznačným vlastnostem jazyka Python patří jeho snadnost co do učení. Python bývá označován za jeden z nejvhodnějších a nejnázornějších programovacích jazyků pro programátorské začátečníky. To je určeno zejména historickým faktem, a sice že jednou z vlivných inspirací při návrhu byl programovací jazyk ABC primárně určený pro výuku a pro použití začátečníky. K názornosti a didaktičnosti programovacího jazyka významně přispívá čistota a jednoduchost syntaxe. Nutno poznamenat, že díky jazyku Python se tím mimo jiné podařilo vyvrátit zažitý úzus, že jazyk určený pro výuku není vhodný pro praxi

a naopak. Důkazem toho je fakt, že python se postupně stále zřetelněji prosazuje jako výukový program na univerzitách jako prostředek k výuce programátorského myšlení a algoritmizace. Jazyk Python byl také navržen tak, aby umožňoval kromě jednoduchých jednoúčelových aplikací také tvorbu velmi rozsáhlých, plnohodnotných aplikací zahrnujících implementaci grafického uživatelského rozhraní (např. wxPython, PySide, PyQT nebo PyGTK).

Velmi zajímavým faktem je, že co do licence je Python vyvíjen jako projekt open-source, a jsou tedy zdarma nabízeny instalační balíky pro takřka všechny běžné operační systémy včetně těch mobilních, tedy pro systémy Unix a Linux, MS Windows, macOS a samozřejmě také pro Android, přičemž pro těsné sepjetí jazyka Python se systémem Linux hovoří fakt, že ve většině distribucí systému GNU/Linux je Python součástí základní instalace. Mimo jiné je v jazyce Python implementován instalátor a většina konfiguračních nástrojů Linuxové distribuce firmy Red Hat.

## 2.2 Python 2.x vs. Python 3.x

Historie a vývoje jazyka Python se týká jedna významná událost, která není ve světě programovacích jazyků vůbec obvyklá. Od svých raných počátků koncem 80. let 20. století prošel jazyk Python návrhem a prvotním vývojem až po uvolnění první číslované komplexnější verze 0.9.0 v roce 1991 následované uvolněním „jedničkové“ verze Python 1.0 v únoru roku 1994, která zahrnovala některé mocné nástroje funkcionálního programování. Následovaly další verze jazyka Python, kde mezi ty významnější z první řady patří 1.2, 1.4, 1.6, to vše stále pod taktovkou původního hlavního autora, nizozemského programátora jménem Guido Van Rossum. Ve druhé generaci tohoto programovacího jazyka, datující se od října 2000 došlo v rámci jednotlivých dílčích verzí v průběhu osmi let ke konsolidaci a celkovému sjednocení struktury jazyka, např. sjednocení datových typů a tříd a uspořádání do pevné hierarchie atd. Ona významná a v zásadě nevídaná zlomová událost ve vývoji Pythonu přišla v prosinci roku 2008 s uvolněním třetí řady jazyka verzí Python 3.0. Zásadní zlom spočívá ve faktu, že nová verze 3.0, respektive celá třetí generace Pythonu, není zpětně kompatibilní[8], a tedy programy napsané v Pythonu předchozí druhé řady obecně nefungují ve verzi 3 a naopak. Ve snaze učinit přechod z jedné verze na druhou poněkud hladší byly verze Python 2.6, která je kompatibilní rovněž s verzí 3.0 a verze Python 2.7 s verzí 3.1, ačkoli kompatibilita zajištěna částečně, resp. zvláštním začleněním stěžejních funkcí. Lze to

[8] PILGRIM, Mark. Ponořme se do Python(u) 3. CZ.NIC, z. s. p. o., 2010. ISBN: 978-80-904248-2-1.

ukázat na příkladu funkce *print*, která se ve druhé generaci Pythonu syntakticky správně zapíše

```
print 1
```

zatímco ve verzi 3.0 a dále vypadá tato funkce takto:

```
print (1)
```

Verze Python 2.6 zvládne přijmout verzi *print (1)*, ačkoli toho dosahuje nutností začlenit tuto funkci z novější verze (importem) na začátku programu prostřednictvím klauzule

```
from __future__ import print_function.
```

Zatímco využití vlastností nebo funkcí Pythonu 3 v Pythonu 2 se řeší importem

```
from __future__ import NázevFuknce
```

existuje i způsob, jímž lze konvertovat Python 2 na Python 3 automaticky. Současně s instalací Pythonu standartně probíhá i instalace nástroje 2to3, který slouží k převodu python2 kódu na python3 kód dle knihovny lit2to3. A to v příkazovém řádku v adresáři Pythonu:

```
2to3 ukazka.py
```

Nutno také dodat, že verzí 2.7 vývoj jazyka Python v rámci druhé generace končí a pokračuje nadále pouze třetí generací. Změn ve verzi Python 3.x oproti předchozí generaci Python 2.x je pochopitelně velmi mnoho, ale pokud jde o to nejpodstatnější a nejvíce viditelné je možné odlišnosti shrnout do několika bodů, které ilustrují elementy skupin změn:

***Unicode kódování*** – zavádí plnou a přirozenou podporu Unicode, např. v řetězcích.

***Změna funkce `print`,*** která se tak stává se změnou syntaxe plnohodnotnou a standardní vnitřní funkcí – viz předchozí příklad.

***Funkce `input` a `raw_input`*** – jde o funkce týkající se uživatelských vstupů z příkazového řádku, konkrétně ve verzi 3.0 je sice zavedena funkce *input*, ale chová se jako funkce *raw_input* ve verzi 2.0, zatímco funkce původní funkce *input* je odstraněna.

***Sjednocení datových typů*** – datové typy *int* a *long* z verze 2 se ve verzi 3 sjednocují pouze na univerzální a jednotný typ *int*.

***Operátor různosti*** – u verze Python 2.x existuje operátor pro různost ve formě <>, zatímco u verze Python 3.x je přípustný pouze operátor *!=*

a mnoho dalších změn, mezi něž patří také přejmenování některých modulů a knihoven, způsob provádění slovníkového testování (hledání výskytu klíče v seznamu), změny u funkcí funkcionálního programování nebo zacházení se standardními výjimkami.

Tento neobvyklý krok pochopitelně vyvolal mnoho rozporuplných reakcí[9] a Pythonu byla na jedné straně předpovídána neslavná budoucnost a brzký konec. Jazyk se tímto krokem také znelíbil mnoha někdejším příznivcům, ovšem na straně druhé bývají změny hodnoceny velmi pozitivně, hovoří se o tom, že programátorovi usnadňují práci a celý - koncepčně již tak přehledný - jazyk ještě více zpřehledňují a činí kód čitelnějším. Složitost celé situace s přechodem z druhé na třetí verzi Pythonu dokumentují také články a komentáře, které se objevují i v roce 2014, tedy šest let po uveřejnění Python 3.x, kdy se Python objevuje již ve verzi 3.4: *„V poslední době se dokonce začínají objevovat nové články o tom, že by se mělo pokračovat s vývojem Pythonu 2, protože Python 3 nepřináší nic převratného a přechod na něj je zbytečně náročný (ať už časově či finančně). To však může vést k roztříštění komunity a není to nejlepší vizitkou pro jazyk jako takový.“*[10]. Nelze jednoznačně říci, kdy je lepší použít Python 2.x a kdy Python 3.x. Vždy záleží na konkrétním projektu, jeho účelu a použitých funkcích a konstrukcích. Nicméně vzhledem k širokému spektru změn je na první pohled zřejmé, že specifika a rozdíly jsou skutečně významné, a to do té míry, že ani základní a jednoduché programy ve verzích 2.x (tedy kromě přechodových verzí 2.6 a zejména 2.7) nejsou pro Python 3.x použitelné beze změna a naopak[11]. Vzhledem k tomu, že podpora Python druhé generace by měla být aktivní až do roku 2020, není ještě starší generace zcela mimo hru. Nicméně už před několika lety se doporučovalo připravovat již běžící projekty založené na starší verzi připravovat na přechod, či tento přechod přímo uskutečnit, a tak se zdá, že v současnosti je na přípravu transformace na Python 3.x nejvyšší čas. Obecně se samozřejmě dá souhlasit se shrnutím, že stávající projekty, u nichž se předpokládá vývoj či podpora i do budoucna, je třeba transformovat na verzi Python 3.x. Pro nové projekty nelze, než doporučit, aby bylo přistoupeno k využití Python 3.X, ideálně v poslední aktuální verzi, a to nejen kvůli technickým vylepšením, ale také s ohledem na kontinuitu vývoje a

[9] STÍSKALA, Viktor. Python 3 může oživit Python. Ale nemyslí si to všichni.
[10] Tamtéž.
[11] PILGRIM, Mark. Ponořme se do Python(u) 3.

podpory. Pro přesnost nutno doplnit, že aktuální verze Pythonu z března roku 2018 nese číslo 3.6.5.

## 2.3 Shrnutí výhod a nevýhod jazyka Python

Výše uvedené vlastnosti jazyka Python lze sumarizovat jako výčet kladů a záporů jazyka.

### 2.3.1 Klady

Mezi objektivní klady jazyka patří zejména s pohledem na téma této práce následující body:

***Snadnost naučit se jazyk*** – dokonce i pro začínající programátory,

***Multiplatformnost*** – s ohledem orientací této práce směrem k OS Android je tato vlastnost zvláště pozitivní,

***Efektivita programování*** – poměrně malé množství kódu potřebného k řešení dané úlohy je bezesporu kladem, spolu se snadnou škálovatelností v případě rozsáhlých aplikací,

***Objektově orientované programování*** – dobrá podpora této moderní a trendové techniky je bezesporu užitečná,

***Mnoho frameworků*** – nejen pro mobilní platformy, ale také pro web, desktopy apod.

***Velká komunita*** – velmi široké rozšíření a z toho plynoucí bohatý komunitní život i velká nabídka hotových řešení,

***Podpora velikánů*** – otevřená podpora gigantů v oblasti IT (Yahoo, Google, IBM, NASA, Nokia atd.) nepřímo svědčí o smysluplnosti a perspektivnosti Pythonu. Dále existuje nespočet aplikací psaných v Pythonu (Dropbox, BitTorrent, Ubuntu Software Center) a nebo jej využívají alespoň pro určitou, například skriptovací, část (Blender, Gimp, Inkscape)

### 2.3.2 Zápory

Žádný jazyk není jen pozitivní bez záporů, ovšem v tomto ohledu platí dvojnásob, že vše záleží na úhlu pohledu a mnohé z nevýhod se s ohledem na konkrétní aplikaci mohou jevit jako nepodstatné. Negativní stránky Pythonu lze shrnout takto:

***Pomalejší provádění*** – při realizaci prací s velkým výpočetním nárokem patří python mezi ty pomalejší,

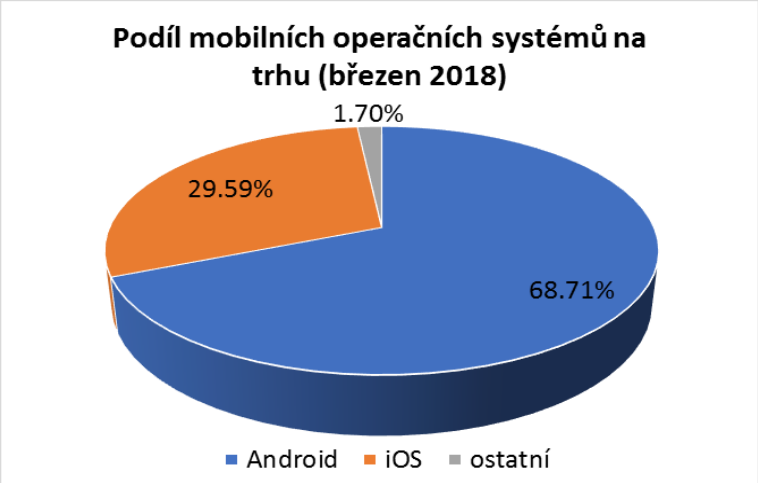
***Úlohy náročné na paměť*** – negativum nepřímo související s předchozím bodem (správa paměti Pythonu není tak pružná a efektivní, jako je tomu např. u jazyků bližších k C, např. Java nebo C++),

***Omezená spolupráce s databázemi*** – Python není přímo stavěný na spolupráci s databázemi, jako je tomu např., v případě webu u PHP,

***Nepříliš vhodný pro víceprocesorové a vícejádrové aplikace*** – opět koresponduje s výše uvedenými kroky, neboť Python nedisponuje tak dokonalou správou multitaskingu a správou paralelních procesů.

# 3 OPERAČNÍ SYSTÉM ANDROID

Operační systém Android, jehož vznik se datuje do r. 2007 a po pouhých zhruba deseti letech své historie představuje dnes zcela evidentně dominantní operační systém v oblasti mobilních zařízení a jeho význam je bezpochyby velmi výrazný. Již na sklonku roku 2010 Android zaujal vedoucí pozici co do zastoupení v sektoru mobilních operačních systémů, s rokem 2012 již přesáhl podíl 59 % a v roce 2013 již byla dominance OS Android zcela jednoznačná. Aktuální stav o drtivě dominantním podílu dvou velkých hráčů na trhu, kterým je kromě OS Android také Apple a jeho iOS, graficky ilustruje koláčový graf na obr. 2.




*Obr. 2: Celosvětový podíl nejznámějších operačních systémů na trhu k období 3/2018 (zdroj: Netmarketshare[12])*

Při aktuálně osmi vydaných hlavních verzích stojí za zmínku charakteristický symbol OS Android, kterým je pojmenovávání po typicky amerických sladkostech, které při jeho verzování používá společnost Google. Dosud vydané verze systému Android s jejich charakteristickými vylepšeními zahrnuje tabulka (Tab. 1).

[12] NETMARKETSHARE. Mobile Operating System Market Share.

*Tab. 1: Přehled verzí OS Android (zdroj: Computerworld[13])*

| *Verze* | *Název* | *Vydání* |
|---|---|---|
| *Android 1.0* | --- | *2008* |
| *Android 1.1* | --- | *2009* |
| *Android 1.5* | *Cupcake* | *2009* |
| *Android 1.6* | *Donut* | *2009* |
| *Android 2.0* | *Eclair* | *2009* |
| *Android 2.2* | *Froyo* | *2011* |
| *Android 2.3* | *Gingerbread* | *2010* |
| *Android 3.0* | *Honeycomb* | *2011* |
| *Android 4.0* | *Ice Cream Sandwich* | *2011* |
| *Android 4.1* | *Jelly Bean* | *2012* |
| *Android 4.4* | *KitKat* | *2013* |
| *Android 5.0* | *Lollipop* | *2014* |
| *Android 6.0* | *Marshmallow* | *2015* |
| *Android 7.0* | *Nougat* | *2016* |
| *Android 8.0* | *Oreo* | *2017* |

## 3.1 Rozdíly v systémech

Podobně jako je tomu u klasických operačních systémů PC jednotlivé verze se při zachování základu struktury a principu fungování přesto mírně liší. Jinak totožné základní verze operačního systému se dle výrobce mobilního zařízení, potažmo dle balíčku aktualizací vzájemně liší. Tím dochází k rozdílům mezi jednotlivými systémy, v jejichž důsledku v rámci dvou různých zařízení s nainstalovanou totožnou verzí OS Android ještě nemusí jít z hlediska funkčnosti aplikací o stejný systém. Výše zmíněné oficiální vydané verze systému se tím dále rozpadají (fragmentují) na další podskupiny. To může z hlediska vývoje aplikací, které by měly na stejné verzi OS na různých zařízeních ideálně fungovat naprosto stejně, způsobit komplikace, které by v rámci vývoje mělo odhalit důsledné testování. Minimální rozhodnutí, které musí vývojář učinit, je rozhodnout se pro verzi operačního systému, pro niž bude aplikaci vyvíjet, pochopitelně chce-li využít plné spektrum funkcí, které se v rámci dané verze nabízejí.

Významným faktorem fragmentace je diferenciace nebo-li rozrůznění zařízení z hlediska vlastností displeje, zejména pak jeho rozlišení. Displej je na klasickém mobilním zařízení nejen jednotkou výstupní, ale kvůli téměř výhradnímu použití dotykových displejů také současně jednotkou vstupní. Rozlišení displeje se specifikuje na základě hustoty pixelů vyjádřené pomocí jednotky dpi (dots per inch), tedy počet pixelů na palec, což může způsobit další komplikace při vývoji zejména graficky orientovaných aplikací či aplikací

[13] Android verze, Computerworld,

s komplikovaným zobrazením s potřebou přesnosti. Operační systém Android z pohledu rozlišení displeje stupňuje mobilní zařízení do šesti skupin[14] a vedle toho na další čtyři skupiny dle rozměru zobrazovací plochy[15].

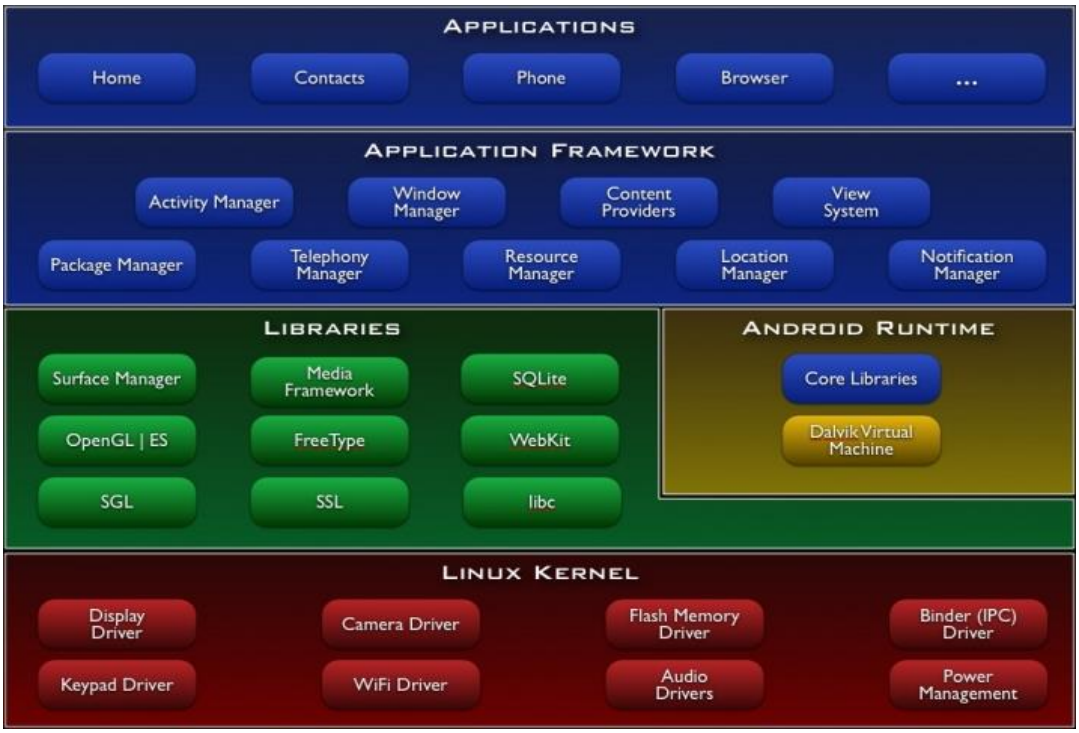
Oběma způsobům výše popsaného rozdělení displejů mobilních zařízení je třeba při vývoji aplikací věnovat zvýšenou pozornost, protože jen tak se bude výsledná aplikace chovat dle požadovaných předpokladů. Na vývojáři pouze spočívá vytvoření sady zdrojových souborů (např. obrázků, rozložení ovládacích prvků obrazovky aplikace apod.) dle výše specifikovaných kategorií a následně ponechat na operačním systému, aby z nabídnutého zvolil tu nejlépe odpovídající variantu. Zřejmým a zásadním cílem je to, aby aplikace vypadala a chovala se shodně na všech typech zařízení.

## 3.2 Architektura operačního systému Android

Operační systém Android patří mezi progresivní, rozsáhlé a komplexní operační systémy s otevřeným zdrojovým kódem (OS Android spadá pode open-source licenci typu Apache License[16]) a je určený primárně jako platforma pro mobilní zařízení. Díky dostupnosti zdrojových kódů umožňuje v zásadě libovolné programátorské úpravy, vývoj komponent a samozřejmě také mobilních aplikací. Android staví na jádře operačního systému Linux ve verzi 2.6, s čímž mimochodem právě koresponduje zmíněná licenční politika. Jádro systému tak zajišťuje zabezpečení systému jako celku, správu pamětí, správu procesů, přístupy k síťovým rozhraním nebo řízení vnitřních komponent a zpracování signálů z interních senzorů. Aplikace jako nadstavba operačního systému pak k jádru nepřistupují přímo, ale prostřednictvím Android API (Application programming interface) rozhraní, tedy rozhraní pro programování aplikací, které lze při programování využívat.

Vlastní architektura operačního systému Android sestává z pětice vrstev, kde každá z nich disponuje množinou operací, které má za úkol provádět více méně samostatně. Z pohledu praxe jsou vrstvy propojené a navzájem spolupracují. Strukturu vrstev OS Android vyjadřuje obr. 3.

[14] Supporting Multiple Screens.
[15] Tamtéž.
[16] Apache License.




*Obr. 3: Struktura vrstev OS Android (zdroj: Android Developers[17])*

***Linux Kernel*** – Nejnižší vrstva architektury na obr. 3 představuje samotné jádro, tedy jádro systému Linux. Jeho základní funkcí je obsluha hardwarových prostředků, tedy v samostatné kategorii stojící řízení napájení a spotřeby zařízení a vedle toho pak hardwarové ovladače pro displej, klávesnici, bezdrátová i kabelová datová rozhraní, sdílené paměti apod. Někdy bývá ještě nad vrstvou jádra zobrazena další vrstva označená Hardware Abstraction Layer (HAL)[18], která má představovat standardizované rozhraní, které zprostředkovává mnohdy značně rozdílně řešené hardwarové prostředky a jejich kapacitu pro vrstvy vyšší. Sestává z množiny knihoven pro specifické skupiny hardwarových prostředků, jako jsou audio zařízení, bezdrátová rozhraní typu bluetooth, obrazové snímací prostředky (kamery, fotoaparáty) apod. Účelem vrstvy je tedy implementace abstrakce mezi použitým hardwarovým zařízením s jeho součástmi a softwarovými entitami ve vyšších vrstvách, potažmo přímo v nejvyšší aplikační vrstvě, která interaguje s uživatelem. Vrstvu HAL si lze představit s určitou mírou abstrakce jako jakousi sadu pomyslných konektorů pro připojení různých hardwarových zařízení.

***Libraries*** – Vrstvu Libraries tvoří knihovny napsané v jazyce C/C++, které využívají různé komponenty systému, a to z toho důvodu, že jazyky rodiny C jsou nativními jazyky jádra Linuxu a veškeré ovladače jsou tedy napsané v tomto jazyce. Knihovny tedy představují možnost přístupu softwarovým nástrojům aplikací k hardwarovým prostředkům skrze jazyky

[17] Platform Architecture.
[18] Tamtéž.

C/C++. Příklady takových množin knihoven jsou naznačeny ve struktuře na obr. X a patří sem např. SSL (Secure Socket Layer) pro obsluhu bezpečné šifrované komunikace, Surface Manager pro obsluhu různých možností vykreslování na displeji, SGL jako prostředek pro zacházení s dvojrozměrnými obrazovými daty (obrázky), WebKit jako open-source engine pro webový prohlížeč nebo Media Framework zprostředkující použití obecných multimediálních prostředků, jako je video či zvuk (např. MPEG4, MP3, AAC atd.).

***Android Runtime*** – Vrstva Android Runtime obsahuje především virtuální stroj DVM (Dalvik Virtual Machine), který je zde nově vytvořen speciálně pro Android zejména z důvodů optimalizace vlastností virtuálního stroje na míru mobilním zařízením (výkon procesoru, správa pamětí a režimu řízení spotřeby atd.). Dle obvyklého scénáře jsou aplikace pro Android naprogramovány v jazyce Java, následně jsou překládané do Java byte kódu a následně do mezikódu pomocí Dalvik kompilátoru. Výsledný kód je pak spuštěn na DVM. Podstatné je, že každá aplikace pak figuruje jako samostatný proces s vlastní instancí DVM.

***Application Framework*** – Vrstva Application Framework je z pohledu vývojáře softwaru tou nejvíce podstatnou. Umožňuje přístup k velkému množství služeb, které mohou být použity přímo v aplikacích a které tak mohou zprostředkovat data v jiných aplikacích, používat hardwarové prostředky zařízení (ve spolupráci se zmíněnými nižšími vrstvami), spouštět dílčí, pomocné či samostatné aplikace na pozadí. Do množiny jednotlivých entit vrstvy patří zejména různé nástroje pro správu, tedy např. Notification Manager pro správu uživatelských hlášení, varování a výzev, dále třeba Activity Manager pro správu životního cyklu aplikace nebo Resource Manager s přístupem k prostředkům grafiky či správě souborů.

***Applications*** – Tuto nejvyšší vrstvu modelu operačního systému Android tvoří základní aplikace, které již využívají běžní uživatelé, a to jak ty, které jsou součástí běžně a nedělitelně integrované výbavy telefonu, jako je správa telefonního seznamu, klient pro SMS či správa nastavení telefonu, dále sem patří přídavné standardně předinstalované výrobcem telefonu, ovšem ne nezbytně nutné, a v neposlední řadě samozřejmě také ty, které si uživatel stáhne a nainstaluje dodatečně na základě svého vlastního rozhodnutí.

# 4 VÝVOJ APLIKACÍ PRO ANDROID V JAZYCE PYTHON

V současnosti je nabídka knihoven pro tvorbu aplikací velmi široká. Přehled nabízených řešení v následujících kapitolách bude zaměřen zejména na možnosti podporující tvorbu grafického uživatelského rozhraní, ačkoli ne třeba doplnit, že v širokém spektru současných možností existují i takové, které jsou zaměřeny spíše na možnosti datové komunikace a sítí, práci s dokumenty XML či především na možnosti multimédií.

## 4.1 Vývojové nástroje v přehledu

Vývojových prostředí umožňujících tvorbu aplikací v jazyce Python pro platformu Android je celá řada. Ačkoli za naprosto nejznámější a zřejmě nejpoužívanější framework je považováno prostředí Kivy, v rámci přehledu je jistě vhodné zmínit se i o dalších zajímavých alternativách.

### 4.1.1 PyQt

Jednu velmi rozšířených a významných alternativ použití jazyka Python pro programování GUI představuje řešení nazvané PyQt. Jedná se o projekt, který představuje aplikační programové rozhraní vytvářející vazbu multiplatformního GUI toolkitu Qt Framework pro programovací jazyk Python. V zásadě se z pohledu Pythonu jedná o plug-in komponentu. Mateřský projekt Qt Framework je velmi rozšířené vývojářské řešení vytvořené v jazyce C++ původně vytvořené firmou Nokia a v současnosti patří ke špičkovým propracovaným vývojovým nástrojům pro grafická aplikační uživatelská rozhraní zejména pro aplikace pro desktopové platformy s podporou všech tří stěžejních operačních systémů (OS X, Linux i MS Windows), ale současně také mobilních platforem Android, iOS i BlackBerry[19]. Vývoj samotného projektu PyQt zaštiťuje britská společnost Riverbank Computing, která PyQt poskytuje podobně jako Qt pod licencí GNU/GPL-2, a také pod placenou licencí. Komponenta PyQt je dostupná pro GNU/Linux a samozřejmě i pro další systémy na bázi Unixu, dále pro macOS a MS Windows.

[19] Struhár 2012


*Obr. 4: Logo PyQt ve dvou používaných verzích (zdroj: Polygon[20]).*

Co do použití je princip tříd i jejich funkcí v zásadě podobný Qt jako takovému, čímž se pod hlavičkou PyQt dosáhne velmi prospěšné kombinace názornosti struktury jazyka Python s mocnými prostředky jazyka C++, ačkoli je třeba brát v potaz některé disproporce mezi přístupem programovacích jazyků Python a C++, jako je třeba rozdílný přístup ke správě paměti nebo zacházení s událostmi a signály. Pro návrh GUI v rámci PyQt se používá nejčastěji prostředí Qt Designer, v němž se GUI aplikace tvoří typicky hierarchickým uspořádáním widgetů, neboli základních funkčních prvků, do oken, skupin či panelů. Pro ilustraci lze uvést jednoduchý příklad konstrukce, jak může být v rámci prostředí Qt Designer vytvořeno grafické rozhraní hlavního okna s tlačítkem a nápisem (label), nastavení jeho vlastností, ukázka použití události stisku tlačítka a rovněž využití dědičnosti objektů:

Nejprve musí být proveden import tříd QtWidgets z PyQt5. To lze provést příkazem:

```
from PyQt5 import QtWidgets
```

Poté je třeba vytvořit aplikace a její hlavní okno, kterému se nastaví atribut Title

```
app = QtWidgets.QApplication([])
main = QtWidgets.QWidget()
main.setWindowTitle('Pokusne PyQt okno')
```

Dále je možno nastavit uspořádání v hlavním okně pomocí některého Layoutu, v této ukázce je zvolen BoxLayout.

```
usporadani = QtWidgets.QHBoxLayout()
main.setLayout(usporadani)
```

V další části kódu je řešeno, jakým způsobem se bude měnit hlášení, při události stistknutí tlačítka. Nejprve je zapotřebí přidat Label s textem do widgetu s nastaveným Layoutem. Následně je vytvořeno a přidáno tlačítko.

```
hlaseni = QtWidgets.QLabel('Hlaseni cislo 1')
usporadani.addWidget(hlaseni)
tlacitko = QtWidgets.QPushButton('Zmena hlaseni')
```

[20] PyQt

```
usporadani.addWidget(tlacitko)
```

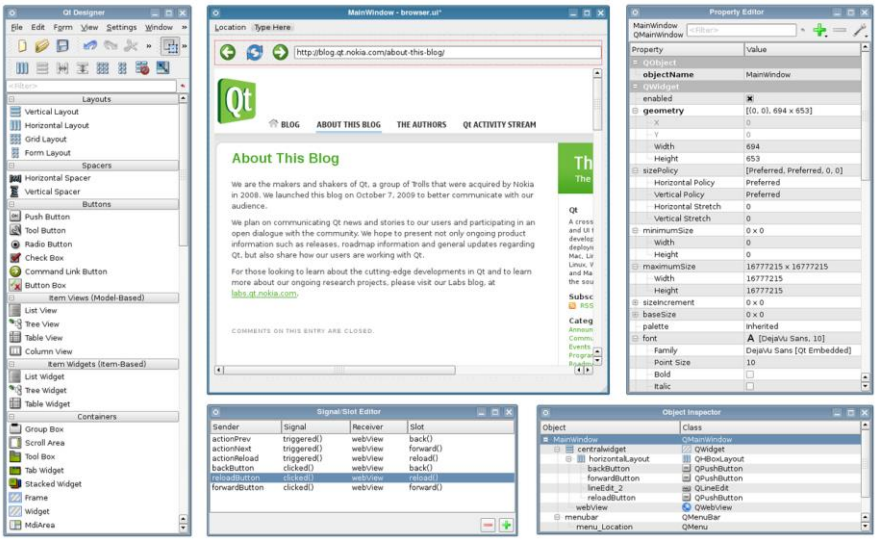
Nyní je třeba nastavit chování tlačítka při stistku, a to vytvořenou funkcí, která je přidána na událost stisknutí tlačítka.

```
def zmena_hlaseni():
    hlaseni.setText('Hlaseni cislo 2')
button.clicked.connect(zmena_hlaseni)
```

A na závěr kód, jenž zajistí spuštění okna a spuštění aplikace s jedním tlačítkem a jedním polem hlášení.

```
main.show()
app.exec()
```

V příkladu výše je také v hlavičce na prvním řádku patrný import knihovny widgetů QtWidgets. Náhled prostředí Qt Designer s připojeným rozšířenm PyQt je na obr. 5.




*Obr. 5: Prostředí Qt Designer (zdroj: Wikiwand[21]).*

Ke kladům frameworku PyQt patří jeho dlouholetá tradice, díky níž je velmi rozvinutá podpora silné a velmi zkušené uživatelské komunity podepřená ještě velmi důslednou oporou firmy Nokia jakožto letité záštity. Velkým přínosem je pak nástroj QtDesigner. Nicméně z hlediska podpory mobilních operačních systémů vyzdvihované klady z hlediska podpory bohužel tak zcela neplatí. Podpora pro vytváření aplikací pro OS Android v rámci tohoto prostředí není zcela nativní, nicméně z PyQt vychází další z hojně využívaných frameworků PySide, kde je situace poněkud lepší.

[21] PyQt Wikiwand

### 4.1.2 PySide

Zejména pro komplikovanou licenční strategii PyQt byl na základě PyQt vytvořen v roce 2009 firmou Nokia toolkit PySide[22] vydaný pod licencí LGPL, která umožňuje použití PySide k vytváření komerčních aplikací, ale také open-source produkty. Podpora PySide co do operačních systémů zahrnuje Linux, OS X, MS Windows a v souvislosti s firmou Nokia také poněkud raritní a již vývojově ukončenou proprietu Maemo. Podpora mobilního OS Android je v přípravě a stále se zlepšuje, přičemž vývoj probíhá komunitním způsobem.


*Obr. 6: Logo toolkitu PySide (zdroj: Root.cz[23]).*

Vzhledem k tomu, že se v případě PySide jedná v zásadě o tentýž přístup k tvorbě aplikací v rámci využití Qt frameworku a základní odlišnost je především v licenčním přístupu, je použití až na detaily totožné, pouze s jinou úvodní klauzulí s importem tříd, jak ostatně ukazuje segment zdrojového kódu z příkladu nastíněného pro PyQt:

Nejdříve je nutno importovat třídy QtWidgets z PySide2:

```
from PySide2 import QtWidgets
```

Poté je již možno vytvořit aplikaci:

```
app = QtWidgets.QApplication([])
```

Dále se vytvoří hlavní okno main, kterému je následně nastaven titulek:

```
main = QtWidgets.QWidget()
main.setWindowTitle('Pokusne okno')
```

A závěrem běžná spouštěcí struktura pro PySide, která zajistí spuštění okna a aplikace:

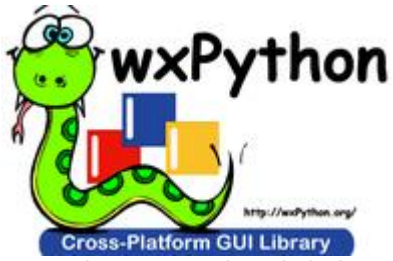
```
main.show()
sys.exit(app.exec_())
```

### 4.1.3 WxPython

Další možností, která dokáže zpřístupnit aplikační programové rozhraní programům psaným v jazyce Python a umožní vývoj aplikací s GUI, je WxPython. Jedná se o multiplatformní

[22] Tvorba grafického uživatelského rozhraní v Pythonu s využitím frameworku PySide.
[23] PySide Root.cz

toolkit, který je co do licencí striktně open-source, tedy volně k použití, a vývoj a úpravy produktu probíhají komunitním způsobem.




*Obr. 7: Logo varianty WxPython (zdroj: WxPython[24]).*

WxPython spadá podobně jako PyQt do robustního a komplexního frameworku WxWidgets pro tvorbu GUI. Jednou ze stěžejních pilířů tohoto prostředí je snaha o co nejlepší přizpůsobení systému, na němž aplikace běží, a to při zachování široké podpory různých operačních systémů včetně starších verzí současných OS, což má být ještě podepřeno velmi důslednou technickou a dokumentační podporou[25]. Podpora multiplatformity jde dokonce tak daleko, že může dojít až k poněkud bizarnímu křížení systémů v tom smyslu, že lze vyvíjet a kompilovat aplikace pro jeden operační systém na běžícím jiném OS (např. vývoj aplikací pro Windows na linuxovém systému s instalací WxWidgets). Multiplatformní charakter prostředí pochopitelně zachovává také propojení WxWidgets s jazykem Python ve formě WxPython[26]. Pokud jde o vývoj na bázi GUI, pro něhož WxWidgets zavádí nástroj zvaný WxGlade, je samozřejmě možné vytvořit propojení (binding) s jazykem Python, čímž je opět, podobně jako u PyQt, dosaženo jednoduchosti a přehlednosti jazyka Python v propracovaném vývojovém prostředí s bohatou nabídkou prvků (widgetů) a jejich použití. Ukázka segmentu zdrojového kódu jednoduché aplikace pro WxPython by mohla vypadat takto:

Nejdříve je nutné zajistit import patřičného balíčku:

```
import wx
```

Následuje vytvoření aplikace a hlavního okna s atributem Title:

```
app = wx.App()
main = wx.Frame(None, title="Pokusne okno")
```

A nakonec část kódu, která zajistí spuštění hlavního okna a aplikace:

```
main.Show()
app.MainLoop()
```

[24] WxPython
[25] Tamtéž.
[26] Tamtéž.

Z již zmíněné stěžejní snahy prostředí WxWidgets přizpůsobit se platformě, na níž běží, a využít maxima nativních prvků plyne jedna z výhod tohoto prostředí, naopak v množině dostupných widgetů poněkud absentuje moderní grafika.

### 4.1.4 Kivy

Framework Kivy patří v současnosti mezi nejrozšířenější vývojová prostředí pro tvorbu aplikací (nejen) pro mobilní operační systémy v jazyce Python. Jedná se o komunitní projekt vyvíjený a šířený pod otevřenou licencí MIT licence[27] (Massachusetts Institute of Technology license zavádí naprosto minimální restrikce co do šíření a nakládání se softwarem) vyvinutý společností Kivy organization poprvé v roce 2011 a v současnosti systém dospěl do verze Kivy 1.10.0. Jedním z typických znaků je jeho multiplatformní koncepce, která v důsledku znamená podporu většiny desktopových i mobilních operačních systémů, a tedy vedle naprosto samozřejmých systémů na bázi Unix/Linux a MS Windows se mezi podporovanými najde i OS X, iOS, Android, ale také např. unikátní jednodeskový počítačový systém Raspberry Pi. Kromě přímé podpory mnoha operačních systémů včetně platformy Android je z hlediska vývoje aplikací pro mobilní zařízení kromě jiného jednou z klíčových vlastností knihovny Kivy také podpora vícedotykového ovládání (multitouch), o jejíž důležitosti svědčí také zdůraznění této vlastnosti hned na titulní straně domovského webu projektu[28] v základní charakteristice Kivy: *„Kivy - Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps.“*.


*Obr. 8: Logo prostředí Kivy (zdroj: Kivy[29]).*

Příklad nejjednodušší aplikace v prostředí Kivy v duchu předchozího příkladu, u něhož je třeba předeslat s ohledem na existenci proprietárního jazyka Kivy, že bude vytvořen s využitím standardního jazyka Python 3, by mohl vypadat zhruba takto:

Nejdřívě je třeba zajistit import balíčku Kivy:

```
import kivy
kivy.require('1.9.0')
```

[27] MIT Licence.
[28] Kivy.
[29] Tamtéž.

Následují importy základních tříd App a Label:

```
from kivy.app import App
from kivy.uix.label import Label
```

Dále přichází vytvoření třídy aplikace:

```
class MyApp(App):
  def build(self):
    return Label(text='Hello world')
```

A nakonec část, která zajistí její spuštnění:

```
if __name__ == '__main__':
MyApp().run()
```

Zřejmě nejvíce multiplatformí přístup vykazuje framework Kivy. Díky poměrně nedávnému vzniku jsou již od základu do Kivy zakomponovány současné moderní grafické prvky. Všechny výše zmíněné systémy patří k těm velmi často používaným. Jelikož alternativních systémů je obrovské množství a byť jen krátce zmínit všechny běžně dostupné vysoce přesahuje rámec této práce, budou za všechny ostatní pro doplnění stručně zmíněny ještě dva systémy, a sice Tkinter a PyGTK jako historicky vysoce významná řešení.

### 4.1.5 Tkinter

Název tkinter (dříve Tkinter) představuje zkráceninu slovního spojení „Tk interface“ a představuje tedy modul, který vytváří aplikační programové rozhraní jazyka Python pro Tk GUI toolkit. Tkinter je otevřený software šířený pod licencí Python License[30], která nepatří k těm nejvolnějším a využití tohoto produktu částečně omezuje. Původní název s velkým písmenem Tkinter byl od Python verze 3 přejmenován na tkinter. Tkinter tvoří v zásadě vrstvu nad grafickou knihovnou Tcl/Tk, což je sada widgetů (základních grafických prvků pro aplikace, např. tlačítka, rámečky, textové popisy, výběrová pole apod.) vytvořenou již v roce 1987. Tkinter nachází oblibu u začínajících programátorů díky své jednoduché implementaci a snadnosti použití, neboť i navzdory tomu při vhodném doplnění prostředí o potřebné moduly lze vytvářet plnohodnotné komerční aplikace. Tk, a potažmo pak i Tkinter je dostupný na všech unixových platformách i na MS Windows. Pro prostředí tkinter by kus zdrojového kódu pro import, vytvoření okna a spuštění aplikace tak, jako v předchozích příkladech, mohl vypadat takto:

Nejdříve je zajištěn import potřebého balíčku:

[30] Python License.

```
import tkinter as tk
```

Poté proběhne vytvoření hlavního (kořenového) widgetu:

```
hlavni = Tk()
```

Dále se vytvoří hlavní okno main s nastaveným atributem Title:

```
hlavni.title('Pokusne okno')
```

A na závěr spuštění okna a aplikace (smyčka událostí)

```
mainloop()
```

### 4.1.6 PyGTK

V případě PyGTK je výchozím prostředím GTK, v jehož případě jde o jeden z nejstarších projektů vytvořený v jazyce C původně pro operační systém Unix. V této souvislosti není bez zajímavosti, že se jedná o prostředí, v němž bylo vytvořeno grafické prostředí Gnome, jedno ze dvou základních původních GUI systémů Unix / Linux. Vývojový systém GTK svými vlastnostmi plně koresponduje se svým původem: pro některá jednoduchá použití může poskytovat výhody nízké úrovně jazyka C (jednoduché úlohy nižších úrovní, např. některé způsoby datové komunikaci či přímá spolupráce s hardwarem řeší jednoduše, nikoli přes vyšší programovací vrstvy), naopak ačkoli GTK disponuje podporou i jiných operačních systému kromě unixových, ovšem tato nemusí být vždy jednoduše realizovatelná bez zvláštních opatření, a ani není příliš patrná tendence podporu jiných OS výrazněji budovat. Pokud jde o binding Pythonu nazvaný PyGTK je situace pochopitelně podobná, tedy hlavní ambice má základ v Linuxu. Vývoj tohoto prostředí však pokračuje jako objektově orientovaný projekt PyGObject pod licencí LGPL. Ukázka zdrojového kódu v duchu předchozích příkladů by mohla mít tuto podobu:

Prvním krokem je import balíčků PyGTK a GTK:

```
import pygtk
pygtk.require('2.0')
import gtk
```

Dalším krokem je vytvoření třídy Base, jež obsahuje konstrukci hlavního okna:

```
class Base:
  def __init__(self):
    self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.window.show()
    def main(self):
        gtk.main()
```

Následuje spuštění hlavního okna:

```
print __name__
if __name__ == "__main__":
    base = Base()
    base.main()
```

## 4.2 Vývojový nástroj Kivy

Kivy jako vysoce moderní nástroj pro tvorbu GUI představuje velmi efektivní prostředek pro vytváření aplikací s přirozeně působícím uživatelským rozhraním na velmi širokém spektru mobilních zařízení. Jeden z důvodů hovořících pro výběr prostředí Kivy, kterým je přímo vestavěná podpora multitouch, byl zmíněn v předchozí kapitole. Kromě toho se ale často uvádějí i další důvody[31], např. že Kivy účelně nahrazuje nepříliš přívětivé API funkce dřívějších grafických rozhraní (např. HTML a CSS), dovoluje připravit tutéž aplikaci pro mnoho operačních systémů najednou atd. K tomu Dusty Phillips ve své publikaci[32] přidává výmluvný důvod pro volbu právě kivy, a sice že *„Kivy je v zásadě jediná životaschopná možnost, jak programovat v pythonu aplikace pro mobilní zařízení.“* (pozn.: volný překlad autor).

Ohledně dvou takřka vedle sebe existujících verzí Python 2 a Python 3, které nejsou navzájem ze znatelné části kompatibilní, existuje z pohledu prostředí Kivy podpora pro obě verze, a to od verze 1.8. Nicméně je ale pochopitelně obecně doporučováno používat pokud možno verzi Python 3 kvůli zjednodušením, efektivitě a také s ohledem na předpokládaný vývoj verzí Pythonu do budoucna. Jednou z velice užitečných specialit Kivy je existence speciálního jazyka KV (KV language), často také nazývaného „kvlang“ s jednoduchou značkovací syntaxí, která se vyznačuje přehledností a čistotou kódu, což staví Kivy v porovnání s ostatními nástroji do zcela výjimečného postavení.

### 4.2.1 Jazyk prostředí Kivy

Prostředí Kivy plně respektuje oddělení prezentační vrstvy aplikací a její funkční úrovně. Pro vytvoření prezentační vrstvy, tedy UI, obsahuje framework Kivy možnost použít jazyk KV někdy tak nazývaný kvlang. Jazyk KV byl vytvořen pro zefektivnění procesu tvorby zejména UI, widgetů atd., pro něž zavádí stromovou strukturu co do nastavení jejich vlastností. Struktura tohoto proprietárního jazyka je směsicí syntaktických pravidel XML,

[31] PHILLIPS, Dusty. Creating Apps in Kivy: Mobile with Python.
[32] Tamtéž.

JSON a Pythonu a zdrojové soubory v jazyce KV mají odpovídající příponu .kv. Příbuznost XML naznačuje značkovací charakter některých konstrukcí, což přidává na přehlednosti kódu a celkově je v tomto řešení kladen velký důraz na jednoduchost zápisu a přehlednou strukturu. Jednoduchost struktury lze reprezentovat na příkladu „ahoj světe“:

Import třídy App z kivy.app

```
from kivy.app import App
Label:
  text: "Ahoj světe"'
```

Vytvoření instance třídy App a zavolání její metody run()

```
App().run()
```

Tato jednoduchý zdrojový kód po spuštění zobrazí černé okno aplikace, kde bude vprostřed plochy vepsán text, který obsahuje řetězec v příslušném parametru třídy `Label`, tedy v tomto konkrétním případě „`Ahoj světe`“.

## 4.3 Příklady tvorby aplikací v prostředí Kivy

V rámci této kapitoly bude proveden ukázkový příklad vytvoření jednoduché aplikace.

### 4.3.1 Instalace framevorku Kivy

Před začátkem práce na vývoji vlastních aplikací je třeba pochopitelně vybavit PC potřebným programovacím prostředím. Pro úplnost je třeba dodat, že pro účely této práce bude k práci v prostředí Kivy využíván PC vybavený operačním systémem MS Windows s těmito základními parametry: Windows 7 Enterprise, Service Pack 1, 64 bitů.

Prvním krokem pro instalaci frameworku Kivy je vybavit PC nejprve instalací programovacího jazyka Python pro příslušný operační systém. Programové prostředí v různých verzích lze stáhnout přímo z domovských stránek Python Foundation[33]. Pro účely této práce byla zvolena instalace Pythonu ve verzi 3.6.5. pro tuto verzi a stejně tak i pro jiné je opět na stránkách Python Foundation k dispozici přehledně zpracovaná dokumentace[34]. Instalaci Pythonu je dobré nasměrovat přímo do kořenového adresáře, pokud je to možné, neboť to může zjednodušit další práci a předejít možným problémům s nastavením cest, a v případě instalace do jiného adresáře, je třeba přidat cestu k němu do registru zvolením možnosti „Add to the PATH“ v prvním kroku při instalaci. Správnost nastavení lze ověřit po instalaci jednoduchým zadáním řetězce pro zjištění verze do příkazového řádku `cmd`.

[33] Python Releases for Window
[34] Tamtéž.

```
python --version
```

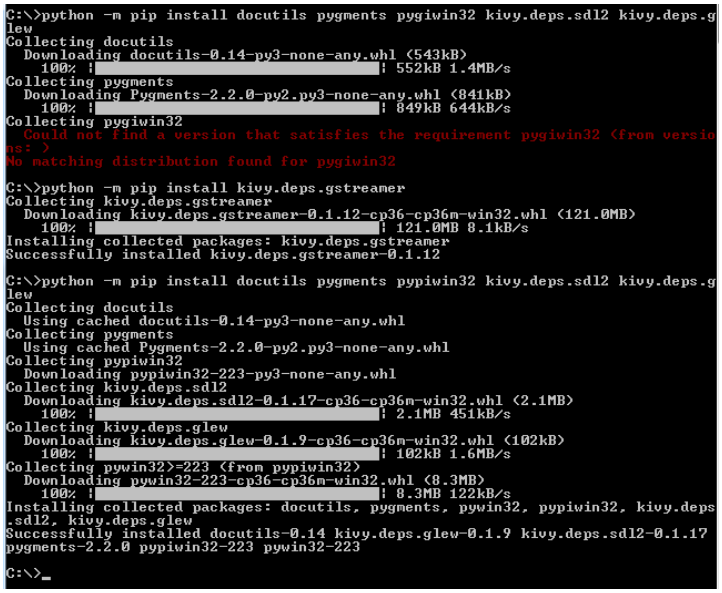
Je třeba také ověřit, zda je instalovaná poslední verze komponent pip a wheel krokem

```
python -m pip install --upgrade pip wheel setuptools
```

Pip je komponenta v roli správce balíčků prostředí Python[35], skrze něhož budou v následujících krocích probíhat koordinované instalace dalších součástí, a wheel představuje řešení zahrnující předpřipravené balíčky komponent s příponou .whl, které lze pak přímo prostřednictvím správce pip instalovat bez nutnosti dalších instalačních kroků. Současně je třeba doinstalovat další potřebné komponenty, které jsou pro korektní provoz požadovány[36], a sice:

```
python -m pip install docutils pygments pypiwin32 kivy.deps.sdl2 kivy.deps.glew
python -m pip install kivy.deps.gstreamer
python -m pip install kivy.deps.angle
```

Nutno doplnit, že dle závislostí je možné zvolit, které komponenty mají být nainstalovány, konkrétně se to týká zejména složky gstreamer (velikost zhruba 120 MB), která souvisí s použitím funkcí audia a videa a nemusí být tedy vždy nutná. Příklad instalace dalších komponent a závislostí ukazuje obr. 9 (obsahuje i překlep při zadávání a opakovanou instalaci a zjišťování závislostí).




*Obr. 9: Příklad instalace prerekvizit prostředí Kivy (zdroj: autor).*

Nyní již lze přikročit k instalaci vlastního prostředí Kivy a příkladů použití (Kivy Examples), které ale nejsou k funkci prostředí v zásadě nutností:

```
python -m pip install kivy
```

[35] User's Guide. Installation on Windows
[36] Tamtéž

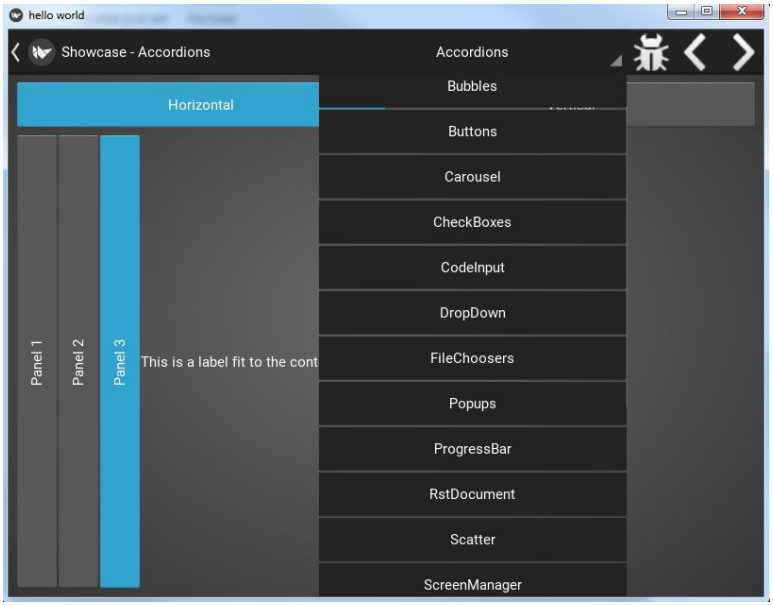
```
python -m pip install kivy_examples
```

Sekvence instalace prostředí Kivy z příkazového řádku ukazuje obr. 10. Pokud tato instalace proběhne bez chyb a v pořádku, je v zásadě hotovo a prostředí Kivy je nainstalováno. Pro ověření funkčnosti prostředí lze spustit ukázkový příklad (cesta zadaná v příkazu vždy závisí na konkrétní instalaci a umístění):

```
python share\kivy-examples\demo\showcase\main.py
```


*Obr. 10: Instalace vlastního prostředí Kivy (zdroj: autor).*

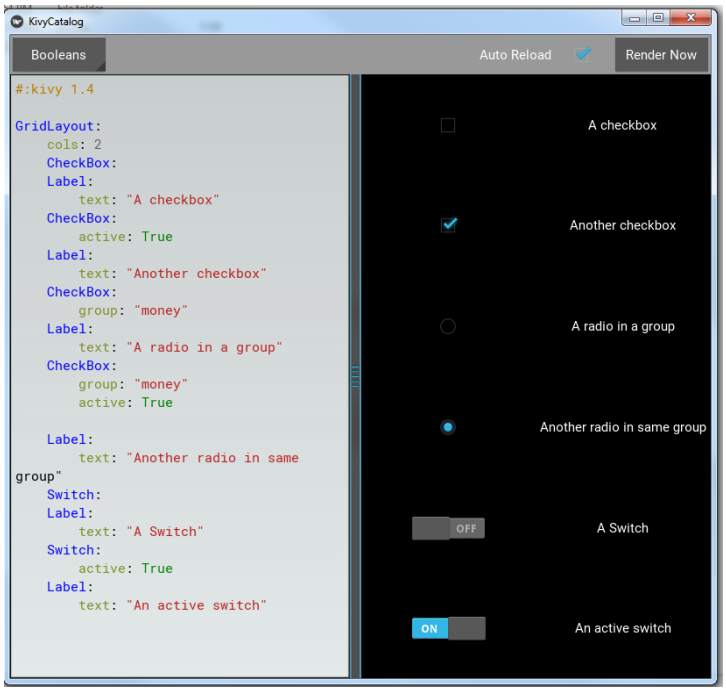
Vestavěný ukázkový demonstrační program prostředí Kivy s předvedením různých uspořádání, zobrazení a ovládacích prvků ukazuje obr. 11. Podobných příkladů je v rámci Kivy Examples k dispozici pochopitelně mnohem více.




*Obr. 11: Jedna z ukázkových demo aplikací prostředí Kivy – ukázka jménem Showcase (zdroj: autor).*

### 4.3.2 Demonstrace použití elementů Kivy

Pro první odzkoušení widgetů a funkcí frameworku Kivy je ideální jedna z demo aplikací, která je pro první pokusy s vlastnostmi a chováním jednotlivých elementů přímo vytvořena. Jedná se o aplikaci nazvanou příznačně KivyCatalog (obr. 12). Aplikace se skládá ze dvou oken, přičemž jedno z nich je určeno pro zobrazení zdrojového kódu ukázky a druhé pak pro vizualizaci (spuštění) tohoto zdrojového kódu. Horní hlavní navigační panel obsahuje v levé navigační rozvinovací roletě sedmnáct položek nabídky, které sdružují skupiny prvků, které tvoří nějaký typický souvislý celek s typickými atributy. Aplikace je možnostem zkoušení uzpůsobena tak, že okno zdrojového kódu umožňuje editaci a jakékoli změny, které jsou ve zdrojovém kódu vytvořeny, se projeví v druhém okně. Lze tak bez potíží použít předvytvořený základ a do něj vkládat vlastní kusy zdrojového kódu v jazyce Kivy a provádět tak jednoduché zkoušky funkčnosti jednotlivých elementů. Tento nástroj může posloužit jako jednoduchá testovací utilita pro odzkoušení částí vlastních aplikací. Samozřejmou možností je editace, či dokonce výměna celých modulů v rámci, neboť v rámci dostupné adresářové struktury lze do této jednoduché aplikace, resp. do jejího rámce, vkládat své vlastní zdrojové soubory přímo kopírováním a přepisováním v rámci adresářové struktury.




*Obr. 12: Jedno z oken editovatelné demo aplikace KivyCatalog (zdroj: autor).*

Nutno doplnit, že užitečnou vlastností je upozornění na chyby v syntaxi příkazů, nicméně online editaci poněkud chybí možnost vkládání textů Ctrl+V. Ovšem pro základní seznámení se strukturou program bohatě stačí a lze jej dokonce bez problémů využít jako základ ke konstrukci vlastních základních aplikací. Další možností, jak vyzkoušet funkčnost prostředí

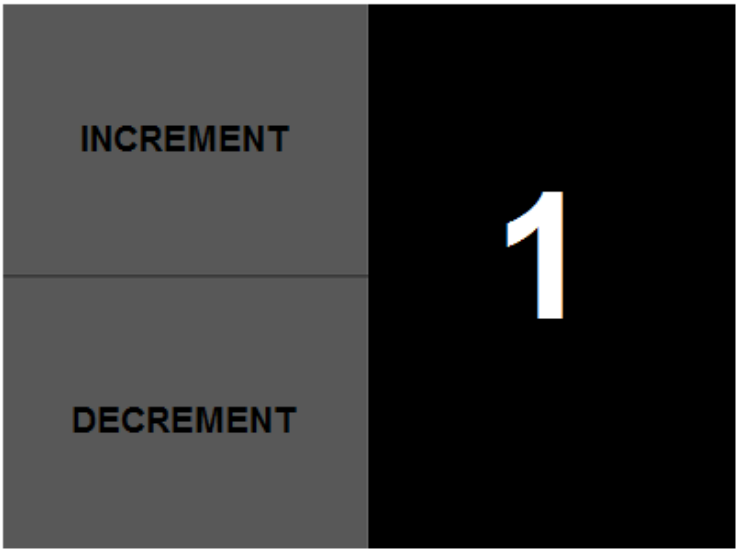
Kivy, zejména pak jeho vazba na prostředí Python, je dobře patrná z tutoriálů (např. jednoduchá hra Pong.

### 4.3.3 Vývoj aplikací s použitím Kivy

V rámci těchto kapitol budou ukázány praktické příklady použití Kivy pro návrh aplikace.

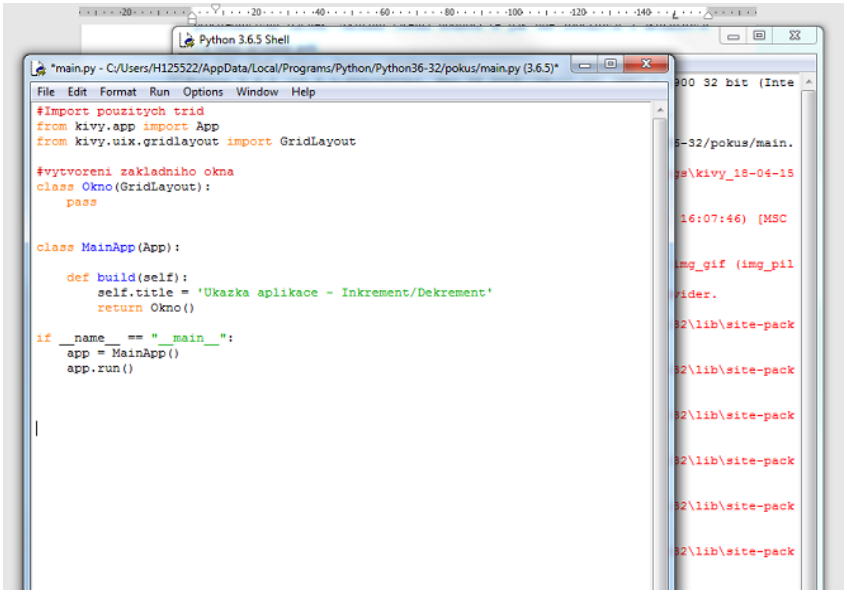
Vytvoření základní aplikace

Základní a nejjednodušší koncept představuje umístění celého UI do jednoho rámce, který se nahraje a spustí ihned po spuštění aplikace, a jeho prvky pak budou představovat jednotlivé widgety. Jako příklad základní aplikace pro demonstraci vlastností a použití prostředí a prvků Kivy bude použita jednoduchá aplikace pro inkrementaci a dekrementaci číselné hodnoty v okně prostřednictvím tlačítek. Aktuální číselná hodnota se pak bude zobrazovat a aktualizovat v k tomu určeném poli.




*Obr. 13: Náhled základu jednoduché demonstrační aplikace (zdroj: autor).*

V prvním kroku je třeba vytvořit a editovat základní soubor `main.py`, který bude rámcem pro základní okno aplikace. Editaci lze provést buď v základním editačním okně prostředí Python (viz obr. 13), nebo bez potíží v jakémkoli textovém souboru a tento soubor pro odladění do vývojového prostředí Pythonu posléze importovat.




*Obr. 14: Náhled základu jednoduché demonstrační aplikace v editoru Pythonu 3.6.5 (zdroj: autor).*

Soubor bude obsahovat základní strukturu potřebnou pro běh aplikace:

Import použitých tříd:

```
from kivy.app import App
from kivy.uix.gridlayout import GridLayout
```

Vytvoření třídy základního okna:

```
class Okno(GridLayout):
    pass
```

Vytvoření třídy aplikace a nastavení jejího názvu:

```
class MainApp(App):

    def build(self):
        self.title = 'Ukazka aplikace - Inkrement/Dekrement'
        return Okno()
```

Klasická struktura pro spuštění aplikace a její běh:

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

Nutno doplnit, že třída GridLayout uspořádává své potomky, kterými budou tlačítka atd. do mřížky s řádky a sloupci.

Dále je třeba vytvořit soubor *main.kv*, který bude představovat, jak ostatně napovídá jeho přípona, soubor v jazyce KV s dalšími součástmi aplikace. Soubor bude obsahovat zatím pouze hrubý základ prázdného okna z upřesněním hlavního kontejneru s prvky aplikace. Soubor main.kv bude obsahovat tyto řádky:

```
<Okno>:
  rows: 1
```

Tlačítka a další prvky obsažené v okně aplikace budou pro zachování přehledné struktury programu umístěny v samostatných souborech ve zvláštním adresáři. Je tedy třeba vytvořit adresář s názvem kv a do něj pak vytvořit další soubor s příponou KV pro vytvoření samotného tlačítka. K tomu je ale třeba v souboru main.py přidat import třídy s tlačítky vytvořit dvě nové třídy pro obsluhu těchto tlačítek:

```
from kivy.uix.button import Button

class PlusOneButton(Button):
    pass

class MinusOneButton(Button):
    pass
```

Do obslužného souboru KV s názvem *buttons.kv* je pak vytvořit samotnou obsluhu tlačítek, resp. metod pro tyto třídy:

```
<PlusOneButton>:
    txt: "Inkrement"

<MinusOneButton>:
    txt: "Dekrement"
```

Je nutno tak říci modulu Kivy, aby z tohoto souboru načítal funkčnosti, resp. aby jej našel v příslušném adresáři. Provede se to v hlavním souboru *main.py* tak, že se parametr předá konstruktoru Builder třeba jednoduše takto:

```
from os import listdir
kv_path = './kv/'
for kv in listdir(kv_path):
    Builder.load_file(kv_path+kv)
```

Zde se importuje třída pro pohyb v adresářové struktuře. Nyní je ještě třeba editovat podobu okna v souboru *main.kv*, který zahrnuje obsah třídy *Okno*. Obsah souboru bude následující (doplněno o popisky):

```
<Okno>:
```

Identifikace místa pro zobrazení:

```
display: display
```

Nastavení struktury do jednoho řádku

```
rows: 1
```

Pro tlacitka je pouzita trida BoxLayout s  vertikalni orientaci prvku

```
BoxLayout:
    orientation: "vertical"
```

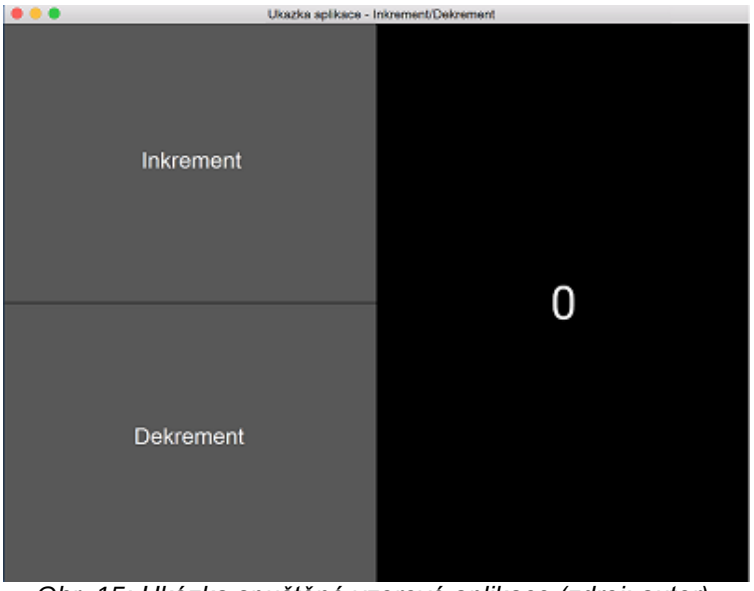
Funkčnost tlačítek v reakci na událost uvolnění tlačítka myší nebo dotekem

```
    PlusOneButton:
        on_release: root.add_one()
    MinusOneButton:
        on_release: root.subtract_one()
```

Obsah třídy Label pro zobrazení hodnoty

```
Label:
    id: display
    font_size: dp(50)
    text: '0'
```

Jak je vidět, tak kromě vizuálních vlastností prvků (např. `font_size`) je zde použita obsluha událostí (reakce na uvolnění tlačítka myši na konkrétních prvcích tlačítek (`on_release`), dále pak definice struktury celého okna. Za pozornost stojí prvek `id` v položce Label, kterého hodnota je `display`. Tato položka v zásadě slouží ke zprostředkování komunikace (předání hodnot) s prostředím Python. Tím je tato jednoduchá aplikace v zásadě hotova. Pro přehlednost je zdrojový kód pro jednotlivé soubory uveden v přílohách takto: `main.py` (Příloha 1), `main.kv` (Příloha 2) a `buttons.kv` (Příloha 2). Ukázka aplikace je na obr. 15.




*Obr. 15: Ukázka spuštěné vzorové aplikace (zdroj: autor).*

### Vytvoření základní aplikace - Canvas

Jako příklad další jednoduché aplikace, která využívá Canvas bude použita ukázka, která při doteku a tažení vykresluje čáru a simuluje tak pohyb prstu, případně tažení myši, po obrazovce. Pro jednoduchost kódu je protentokrát vynecháno rozložení pomocí Layoutu a prvky kv languge. K vyčištění plátna slouží tlačítko VYMAZAT, o ukončení aplikace se pak stará tlačítko ZAVŘÍT. Obě tlačítka jsou umístěna staticky.


*Obrázek 16: Ukázka spuštěné Canvas aplikace (zdroj: autor).*

Soubor bude obsahovat následující strukturu potřebnou pro běh aplikace:

Nejprve je za potřebí importovat využívané třídy, které jsou nezbytné pro její běh. Import tříd probíhá na následujících řádkách.

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

Následně se vytvoří třída pro Canvas, jež se stará o vytvoření plátna Canvas a určitě chování při doteku, případně posunu po plátně. To znamená, že při doteku se začne vykreslovat čára z místa doteku pevně danou barvou RGB modelu.

```
class CanvasTestWidget(Widget):
    def on_touch_down(self, touch):
        with self.canvas:
            color = (1, 1, 0)
            touch.ud['line'] = Line(points=(touch.x, touch.y))

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

Nyní je možno vytvořit funkce, které se v navazující části kódu budou využívat díky stitsku tlačítek pro základní ovládání aplikace. V tomto případě se jedná o funkci, která zajistí vyčištění plátna a funkci, jež ukončí aplikaci.

```
    def clear_canvas(self, obj):
        self.painter.canvas.clear()
    def close_app(self, obj):
        CanvasTest.stop(self)
```

Dalším potřebným krokem je vytvoření třídy aplikace. Zde se nejprve nastaví název aplikace, poté se volá výše vytvořená třída. Dále se v této třídě vytvoří komponenty tlačítek, kterým je následně nastaveno dané chování (funkce) při stistknutí. Veškeré vytvořené komponenty je poté za potřebí přidat do již vytvořeného Widgetu.

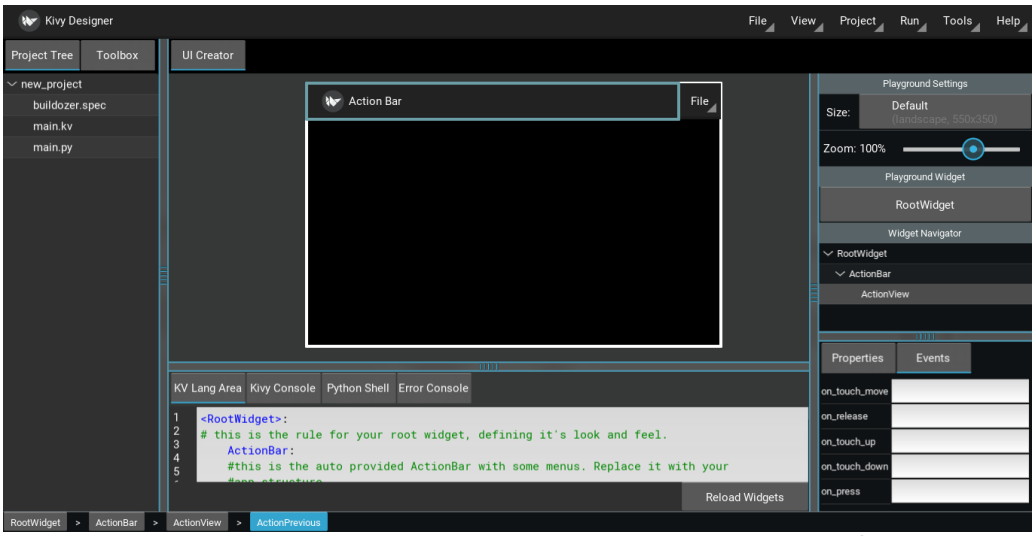
```
class CanvasTest(App):

    def build(self):
        self.title = 'Ukazka aplikace - Canvas'
        parent = Widget()
        self.painter = CanvasTestWidget()
        clearbtn = Button(text='VYMAZAT')
        closebtn = Button(text='ZAVŘÍT',pos=(100,0))
        clearbtn.bind(on_release=self.clear_canvas)
        closebtn.bind(on_release=self.close_app)
        parent.add_widget(self.painter)
        parent.add_widget(clearbtn)
        parent.add_widget(closebtn)
        return parent
```

Klasická struktura pro spuštění aplikace a její běh:

```
if __name__ == '__main__':
    CanvasTest().run()
```

Pro přehlednost je zdrojový kód rovněž uveden v příloze Příloha C – soubor main.py

Pro modelové příklady byla pro názornost zvolena editace kódu přímo v textové podobě v prostředí IDLE Python. Nutno nicméně zmínit další možnost řešení navrhované aplikace, která by se v praxi uplatnila zejména v případě rozsáhlejšího problému s mnoha prvky, a sice KivyDesigner. Zde by návrh této aplikace probíhal v prostředí, jehož náhled je na obr. 16.




*Obr. 17: Okno prostředí Kivy Designer (zdroj: Kivy Designer[37])*

Vprostřed okna aplikace je okno návrhu UI, v levém okně se pak zobrazuje stromová struktura projektu a dole pod hlavním oknem UI designu je pak okno se zdrojovým kódem programu. Výhoda tohoto způsobu řešení spočívá především v tom, že jsou přímo názorně vidět úpravy. Kromě těchto tří základních zobrazovacích ploch je k dispozici také výpis chybových hlášení, nabídka widgetů atd. Nutno zmínit i nevýhodu KivyDesigneru – je nestabilní a dokonce ani vývojáři na svém oficiálním webu jeho využití začátečníkům příliš nedoporučují.

[37] Kivy Designer. Quick-start.

# ZÁVĚR

Cílem této bakalářské práce bylo zvolit vhodné technické programovací prostředky pro tvorbu mobilní aplikace pro operační systém Android. Z mnoha v práci zmíněných důvodů, jako je programátorská efektivita, nativní prostředí nebo přehlednost kódu, byl jako programovací jazyk vybrán Python ve verzi 3, konkrétně verze 3.6.5. Dále podstata práce spočívala ve zmapování nabídky vývojových modulů vhodných k propojení právě s jazykem Python. Jednotlivé varianty byly podrobněji popsány a posouzeny z hlediska jejich možností zejména pokud jde o možnosti tvorby aplikací pro mobilní operační systém Android. Z provedeného výběru a zmapování dostupných řešení bylo pro tvorbu vzorové aplikace zvoleno řešení Kivy, které je v současné době pro návrh aplikací s propracovaným uživatelským rozhraním jedním z nejvyužívanějších. Na základě zvoleného řešení bylo provedeno zprovoznění systému Python s nadstavbou Kivy a na tomto prostředí byla pak demonstrována tvorba aplikace na jednouchých příkladech, které na poměrně snadno čitelném příkladu demonstrují obvyklou strukturu programu, způsob napojení modulu Kivy na základní prostředí Python a také nastavení jednotlivých vlastností použitých prvků. Jedním ze zásadních rozhodnutí, které bylo třeba během řešení tématu řešit, byla volba verze jazyka Python. K použití programování na mobilních platformách se ještě donedávna používal téměř výhradně Python verze 2.x. Nicméně vzhledem k tomu, že podpora Python druhé generace končí v roce 2020 se obecně doporučuje pro (aktivní) projekty založené na starší verzi přechod na verzi 3.x. Pro nové projekty je pak třeba přistoupit k využití Python 3.x rovnou od počátku vývoje.

Z obsahu práce je patrné, že aplikace pro operační systém Android je možné vytvářet i jinými jazyky, které nejsou pro vývoj Android aplikací zcela obvyklé. A právě pro lidi, kteří již ovládají Python, chtějí vytvářet android aplikace a zároveň se chtějí vyhnout výuce dalšího programovacího jazyka (například Java), byť za ním stojí větší komunitní podpora, by mohla být tato práce inspirací a přínosem.

# RESUMÉ

Hlavním cílem této bakalářské práce bylo zjistit, jakým alternativním způsobem lze vyvíjet aplikace pro operační systém Android, a to přesněji v jazyce Python s využitím jeho modulů.

Po zmapování a porovnání dostupných modulů byl zvolen ten, jež byl považován za nejvhodnější, konkrétně se jedná o modul Kivy.

S pomocí vybraného modulu byly v jazyce Python vytvořeny dvě jednoduché demonstrační aplikace, které dokazují, že existují alternativní efektivní způsoby pro vývoj mobilních aplikací pro operační systém Android.

# SUMMARY

The main goal of this bachelor thesis was to find out in what way it is possible to develop applications for the Android operating system, more precisely in Python using its frameworks.

After mapping and comparing the available frameworks, the one that was considered the most appropriate was chosen, specifically the Kivy module.

Using the selected module, two sample applications have been developed in Python to demonstrate that there are alternative effective ways how to develop mobile applications for the Android operating system.

# SEZNAM LITERATURY A INFORMAČNÍCH ZDROJŮ


Android verze. [online]. Computerworld, 10. 5. 2018 [cit. 2018-06-17]. Dostupné z Internetu: https://www.computerworld.com/article/3235946/android/android-versions-a-living-history-from-1-0-to-today.html

ANDROIDCENTRAL. Android's early days. [online]. Android Central [cit. 2018-04-07]. Dostupné z Internetu: http://www.androidcentral.com/androids-early-days

Apache License. [online]. Apache, 17. 3. 2018 [cit. 2018-03-21]. Dostupné z Internetu: http://www.apache.org/licenses/

ELGIN, Ben. Google Buys Android for Its Mobile Arsenal. [online]. Bloomberg L.P., 2005-08-17 [cit. 2011-05-02]. Dostupné na Internetu: http://webcitation.org/5wk7sIvVb

History of Nokia part 2: Snake. In: Converzations by Nokia [online]. 20.1.2009 [cit. 2018-04-06]. Dostupné z: http://conversations.nokia.com/2009/01/20/history-of-nokia-part-2-snake/

Kivy Designer. Quick-start. [online]. Kivy Designer [cit. 2018-04-05]. Dostupné z Internetu: http://kivy-designer.readthedocs.io/en/latest/quickstart.html

LONG, James P. Complete Guide For Python Programming: Quick & Easy Guide To Learn Python. CreateSpace Independent Publishing Platform, 2015. 252 stran. ISBN 978-1-511-84774-2.

LUTZ, Mark. Python Pocket Reference: Python In Your Pocket. O'Reilly Media, 2014. 266 stran. ISBN 978-1-449-35701-6.

LUTZ, Mark. Learning Python. O'Reilly Media, 2013. 1648 stran. ISBN: 978-1-449-35573-9.

MIT Licence. [online]. Choosealicense, 20. 3. 2018 [cit. 2018-03-25]. Dostupné z Internetu: https://choosealicense.com/licenses/mit/

.NET application architecture guide. 2nd ed. Redmond, Wash.: Microsoft, c2009. Patterns & practices. ISBN 978-0735627109.

NETMARKETSHARE. Mobile Operating System Market Share. [online]. Netmarketshare, 25. 3. 2018 [cit. 2018-03-25]. Dostupné z Internetu: https://netmarketshare.com/operating-system-market-share.aspx

Nokia 9210 Communicator. [online]. Nokia, 23. 8. 2013 [cit. 2018-03-25]. Dostupné z Internetu: http://press.nokia.com/2000/11/21/the-nokia-9210-communicator-heralds-the-dawn-of-mobile-multimedia/

Overview of wxPython. [online]. WxPython, 2018 [cit. 2018-04-11]. Dostupné z Internetu: https://wxpython.org/pages/overview/

PHILLIPS, Dusty. Creating Apps in Kivy: Mobile with Python. O'Reilly Media, 2014. 188 stran. ISBN-13: 978-1491946671.

Platform Architecture. [online]. Android Developers, 21. 3. 2018 [cit. 2018-03-21]. Dostupné z Internetu: https://developer.android.com/guide/platform/index.html

Python Releases for Windows. [online]. Python, 2018 [cit. 2018-04-08]. Dostupné z Internetu: https://www.python.org/downloads/windows/

STÍSKALA, Viktor. Python 3 může oživit Python. Ale nemyslí si to všichni. Zdrojak.cz, 30.5.2014 https://www.zdrojak.cz/clanky/python-3-muze-ozivit-python/

Supporting Multiple Screens. [online]. Android Developers, 25. 3. 2018 [cit. 20178-03-25]. Dostupné z: https://developer.android.com/guide/practices/screens_support.html

SUMMERFIELD, Mark. Programming in Python 3: A Complete Introduction to the Python Language. Addison-Wesley Professional, 2009. ISBN 978-0-321-68056-3.

Symbian: příběh nejrozšířenějšího OS [online]. Mobilenet, 29. 9. 2009 [cit. 2018-03-25]. Dostupné z Internetu: https://mobilenet.cz/clanky/pehled-verz-symbianu-funkce-a-vylepen-5877

Tvorba grafického uživatelského rozhraní v Pythonu s využitím frameworku PySide. [online]. Root, 28. 11. 2017 [cit. 2018-03-29]. Dostupné z Internetu: https://www.root.cz/clanky/tvorba-grafickeho-uzivatelskeho-rozhrani-v-pythonu-s-vyuzitim-frameworku-pyside/

ULLOA, Roberto. Kivy: Interactive Applications in Python. Packt Publishing, 2015. 157 stran. ISBN: 978-1-78528-692-6.

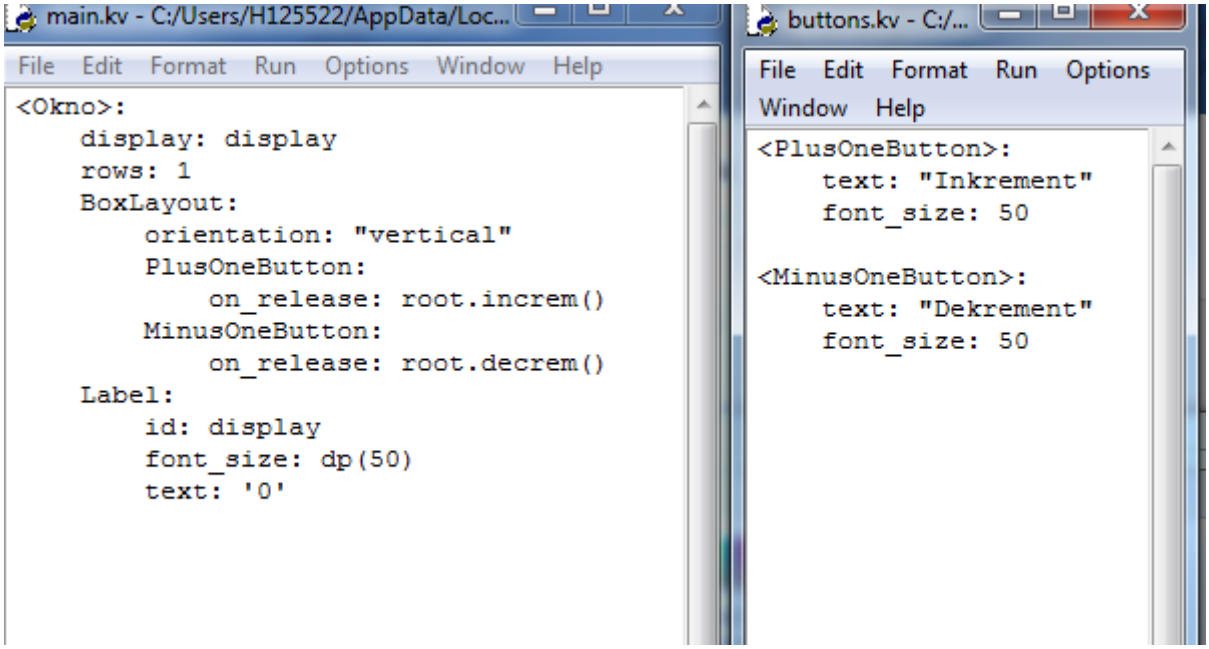
User's Guide. Installation on Windows. [online]. Kivy, 2018 [cit. 2018-04-10]. Dostupné z Internetu: https://kivy.org/docs/installation/installation-windows.html

# PŘÍLOHY

**Příloha A – soubor main.py**

```
#Import pouzitych trid
from kivy.app import App
from kivy.lang import Builder
from kivy.uix.button import Button
from kivy.properties import ObjectProperty
from kivy.uix.gridlayout import GridLayout

#Nastaveni adresare pro zdrojovy soubor tlacitek
from os import listdir
kv_path = './kv/'
for kv in listdir(kv_path):
    Builder.load_file(kv_path+kv)

class PlusOneButton(Button):
    pass

class MinusOneButton(Button):
    pass

#Vytvoreni tridy zakladniho okna
class Okno(GridLayout):
    display = ObjectProperty()

    # Funkce inkrementace
    def increm(self):
        value = int(self.display.text) #prevod textu na INT
        self.display.text = str(value+1)#vlastni inkrementace + TXT

    # Funkce dekrementace
    def decrem(self):
        value = int(self.display.text)#prevod textu na INT
        self.display.text = str(value-1)#vlastni dekrementace + TXT

#Vytvoreni tridy aplikace a nastaveni jejiho nazvu
#s konstruktorem build
class MainApp(App):

    def build(self):
        self.title = 'Ukazka aplikace - Inkrement/Dekrement'
        return Okno()

#Klasicka struktura pro spusteni a beh aplikace
if __name__ == "__main__":
    app = MainApp()
    app.run()
```

**Příloha B – soubor main.kv a buttons.kv**



**Příloha C – soubor main.py**

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


class MyPaintWidget(Widget):

    def on_touch_down(self, touch):
        with self.canvas:
            color = (1, 1, 0)
            touch.ud['line'] = Line(points=(touch.x, touch.y))

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


class MyPaintApp(App):

    def build(self):
        self.title = 'Ukazka aplikace - Canvas'
        parent = Widget()
        self.painter = MyPaintWidget()
        clearbtn = Button(text='VYMAZAT')
        closebtn = Button(text='ZAVŘÍT',pos=(100,0))
        clearbtn.bind(on_release=self.clear_canvas)
        closebtn.bind(on_release=self.close_app)
        parent.add_widget(self.painter)
        parent.add_widget(clearbtn)
        parent.add_widget(closebtn)
        return parent

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

    def close_app(self, obj):
        MyPaintApp.stop(self)


if __name__ == '__main__':
    MyPaintApp().run()
```