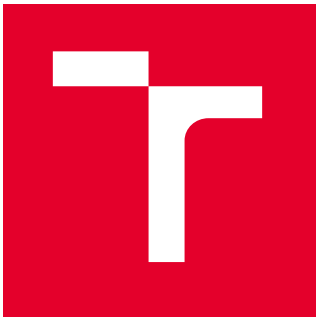


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

PYTHON SKRIPTY PRO ÚPRAVU PDF DOKUMENTŮ

PYTHON SCRIPTS FOR AUTOMATED EDITING OF PDF FILES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Patrik Škeřík**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Pavel Hanák, Ph.D.**

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Patrik Škeřík ***ID:*** 240982

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Python skripty pro úpravu PDF dokumentů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vnitřní strukturou PDF dokumentů, zejména s typy fontů, jejich kódováním a ToUnicode CMap tabulkami. Následně vytvořte skripty, které budou schopny detekovat chybějící ToUnicode mapování a u vybraných fontů jej doplnit. K tomu využijte buď uživatelem vytvořené korekční tabulky nebo tabulky extrahované ze zabudovaných fontů. Skripty musejí umět automaticky či polo-automaticky zpracovat větší množství podobných PDF dokumentů a uživatele upozornit na chyby, které při tom mohou vzniknout. Jejich funkci ověřte nejméně na 20 vzorcích, které vám poskytne vedoucí. Korekční tabulky a konfigurační soubory musejí být koncipovány tak, aby je bylo možné snadno adaptovat na jiné dokumenty. Pokud to bude možné, tak při tvorbě využijte pouze open-source softwarové balíky jako PDFminer.six, PikePDF, MuPDF nebo FontForge. Skripty opatřete podrobnými komentáři funkce a návodem k použití.

DOPORUČENÁ LITERATURA:

[1] Adobe Systems Incorporated, Adobe CMap and CIDFont Files Specification, 1993. Dostupné online https://adobe-type-tools.github.io/font-tech-notes/pdfs/5014.CIDFont_Spec.pdf

[2] Summerfield, M. Python 3. Computer Press, 2021, ISBN 978-80-251-5030-6

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: Ing. Pavel Hanák, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Předmětem bakalářské práce je vytvoření skriptu v jazyce python s použitím knihovny pikePDF, který opraví nesprávné kódování znaků v PDF dokumentech.

KLÍČOVÁ SLOVA

PDF, pikePDF, python, Unicode, CID, glyfy

ABSTRACT

The subject of the Bachelor thesis is programming a script in python programming language using the pikePDF library. This script fixes incorrect encoding of symbols in PDF documents.

KEYWORDS

PDF, pikePDF, python, Unicode, CID, glyfs

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

ŠKEŘÍK, Patrik. *Python skripty pro úpravu PDF dokumentů*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: Ing. Pavel Hanák, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Patrik Škeřík
VUT ID autora:	240982
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Python skripty pro úpravu PDF dokumentů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Pavlovi Hanákovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato bakalářská práce se zaměřuje na problematiku špatného kódování PDF dokumentů. Chybné kódování má za následek nemožnost efektivního vyhledávání v dokumentu a nepřesný přenos textu při kopírování.

Hlavním cílem této práce je prezentovat a implementovat nástroj pro opravu kódování PDF dokumentů pomocí Python skriptu a open-source knihoven, jako jsou `PDFminer.six`, `PikePDF` a `MuPDF`. Řešení těchto nedostatků bude provedeno s ohledem na specifické potřeby časopisu „Amatérské rádio“ od společnosti AMARO. Důvodem nesprávného kódování konkrétně v těchto časopisech je, že k vysázení časopisů se používá desktop publishingová aplikace `Adobe PageMaker`, která není vyvíjena od roku 1999 a je kompatibilní pouze s operačními systémy `Windows XP` a níže. V průběhu této práce budou analyzovány metody opravy kódování a bude provedeno jejich nasazení na konkrétní příklady. Celkovým záměrem je přispět k zlepšení kvality digitálních dokumentů a usnadnit uživatelům práci s elektronickým obsahem různých PDF dokumentů. Výsledně vytvořený skript bude zveřejněn jako open-source projekt, což umožní ostatním uživatelům opravit své vlastní digitální kopie.

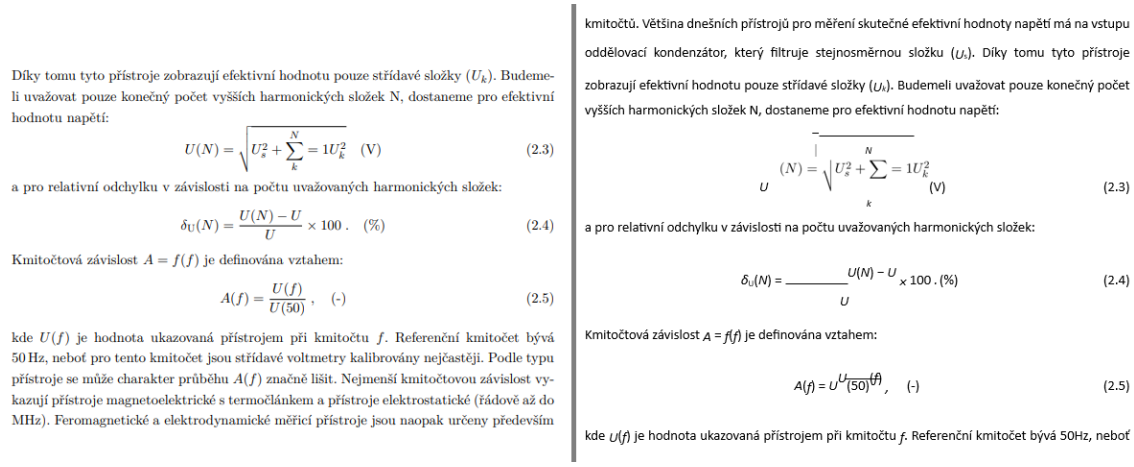
V první části práce se zaměří na samotný formát PDF a jeho kódování. Ve druhé části bude hovořit o návrhu a realizaci řešení.

1 Co je PDF?

PDF (Portable Document Format) je formát souborů vynalezený společností Adobe v roce 1993. Cílem projektu bylo vytvořit formát, který umožní ukládání, posílání, prohlížení a tisknutí dokumentu na jakémkoli zařízení (tedy portable), které formát PDF podporuje [1]. Soubor PDF obsahuje všechny informace potřebné k zobrazení dokumentu v totožné podobě, ve které byl uložen autorem, to zahrnuje přesnou pozici obrázků a tabulek, ale také fonty, které nejsou v zařízení nainstalovány. PDF podporuje nejen vektorovou ale i bitmapovou grafiku.

Před vynálezem PDF bylo nekonzistentní renderování dokumentů mnohem větší problém než dnes, částečně kvůli velkému množství aplikací pokročilého zpracování textu (wordprocessing) a desktop publishing aplikací, které byly dostupné. Toto byl významný důvod pro koupi pro širokou veřejnost.

V roce 2008 se PDF stalo otevřeným standardem ISO 32000-1:2008 [2] což umožnilo ostatním společnostem lepší kompatibilitu s tímto formátem. Společnost Adobe měla však 15 let zkušeností s PDF navíc, proto převod PDF dokumentů do například .docx formátu pomocí microsoft word není dokonalé a stále nastávají chyby jak je vidět na obr. 1.1.



Díky tomu tyto přístroje zobrazují efektivní hodnotu pouze střídavé složky (U_k). Budeme-li uvažovat pouze konečný počet vyšších harmonických složek N, dostaneme pro efektivní hodnotu napětí:

$$U(N) = \sqrt{U_s^2 + \sum_k^N = 1U_k^2} \quad \text{(V)} \tag{2.3}$$

a pro relativní odchylku v závislosti na počtu uvažovaných harmonických složek:

$$\delta_U(N) = \frac{U(N) - U}{U} \times 100\,. \quad (\%) \tag{2.4}$$

Kmitočtová závislost $A = f(f)$ je definována vztahem:

$$A(f) = \frac{U(f)}{U(50)}\,, \quad (\text{-}) \tag{2.5}$$

kde $U(f)$ je hodnota ukazovaná přístrojem při kmitočtu f. Referenční kmitočet bývá 50 Hz, neboť pro tento kmitočet jsou střídavé voltmetry kalibrovány nejčastěji. Podle typu přístroje se může charakter průběhu $A(f)$ značně lišit. Nejmenší kmitočtovou závislost vykazují přístroje magnetoelektrické s termočlánkem a přístroje elektrostatické (řádově až do MHz). Feromagnetické a elektrodynamické měřicí přístroje jsou naopak určeny především

kmitočtů. Většina dnešních přístrojů pro měření skutečné efektivní hodnoty napětí má na vstupu oddělovací kondenzátor, který filtruje stejnosměrnou složku (U_s). Díky tomu tyto přístroje zobrazují efektivní hodnotu pouze střídavé složky (U_k). Budemeli uvažovat pouze konečný počet vyšších harmonických složek N, dostaneme pro efektivní hodnotu napětí:

$$U(N) = \sqrt{U_s^2 + \sum_k^N = 1U_k^2} \quad \text{(V)} \tag{2.3}$$

a pro relativní odchylku v závislosti na počtu uvažovaných harmonických složek:

$$\delta_U(N) = \frac{U(N) - U}{U} \times 100\,. \quad (\%) \tag{2.4}$$

Kmitočtová závislost $A = f(f)$ je definována vztahem:

$$A(f) = \frac{U(f)}{U(50)}\,, \quad (\text{-}) \tag{2.5}$$

kde $U(f)$ je hodnota ukazovaná přístrojem při kmitočtu f. Referenční kmitočet bývá 50Hz, neboť

Obr. 1.1: vzhled dokumentu po převedení z .pdf do .docx pomocí microsoft word (vlevo .pdf, vpravo .docx)

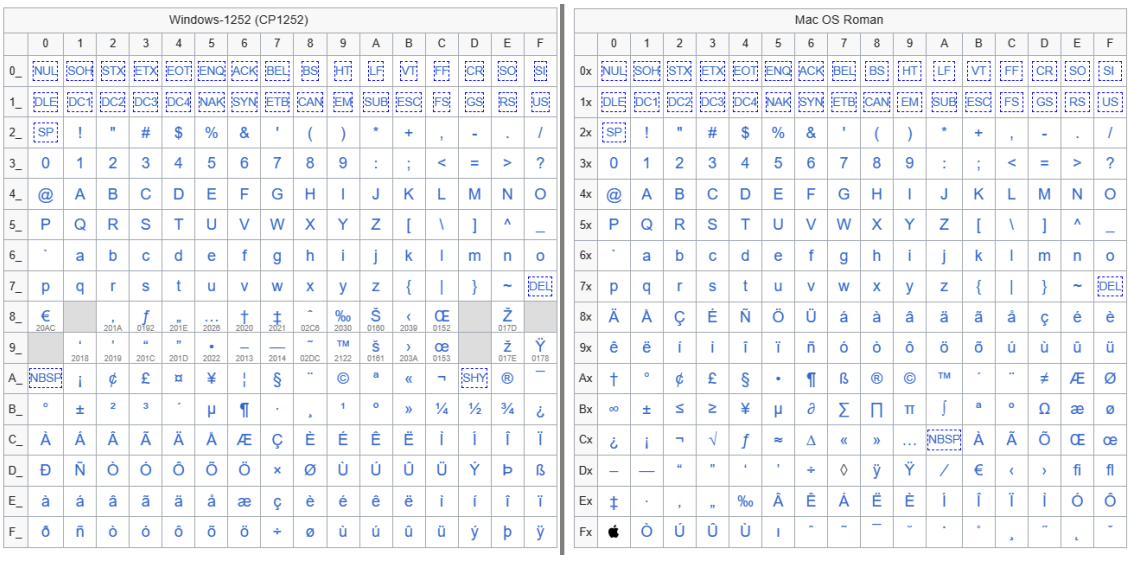
1.1 PostScript a PDF

PostScript je programovací jazyk vytvořený firmou Adobe Systems poprvé zveřejněný v roce 1984. Sloužil jako programovací jazyk pro popis stránky a grafických

objektů [3]. Původně byl vytvořen k tisku dokumentů na laserových tiskárnách. PostScript umožňuje popsat, jak má být stránka nebo obrázek vytisknut na tiskárně nebo zobrazen na obrazovce. Formát PDF byl vytvořen později a byl silně inspirovaný PostScriptem. Dodnes je možné použít PostScript kód vnořený do PDF. To znamená že PDF může obsahovat instrukce v jazyce PostScipt, což umožňuje podrobnější popis grafických objektů a stránek. Jedním z důvodů proč PDF vznikl je tato možnost kombinovat PostScript s dalšími funkcemi jako například metadata, šifrování a vložení interaktivních prvků.

1.2 Kódování v PDF souborech

V PDF se používají kódy znaků k zobrazení v textovém řetězci, které jsou poté mapovány na viditelné symboly pomocí kódování v závislosti na aktuálním fontu. K tomu se využívají kódovací standardy. Mezi tyto standardy patří WinAnsi, MacRoman, MacExpert, Standard encoding, PDFDocEncoding, Identity-(H/V), CJK encoding a kódování vestavěné ve fontu. Různé standardy mají odlišné mapování kódů znaků na symboly např. WinAnsi (Windows-1252) zobrazí znak 0xAE jako ®, kdežto MacRoman (Mac OS Roman) zobrazí znak se stejným číslem jako Æ. Porovnání tabulek těchto standardů můžeme vidět na obr. 1.2.



Obr. 1.2: Porovnání tabulek WinAnsi (Windows-1252 (CP1252)) a MacRoman (Mac OS Roman)

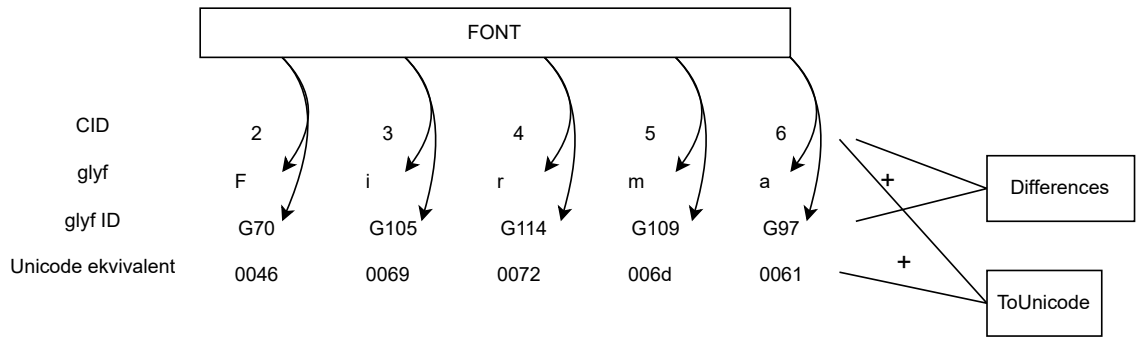
V kódování se každý symbol v řetězci může zapsat jako glyf (např. ve tvaru „G70“ který ve fontu `arial` odpovídá písmenu „F“), ke kterému se přiřadí CID (neboli

character identification, tedy indentifikace symbolu). CID často začíná od čísla dvě a postupně se inkrementuje s každým dalším glyfem použitým v dokumentu. Pokud se písmeno vyskytuje v textu vícekrát, nepřiřazuje se mu nové CID. Po přiřazení glyfů a CID se může použít například `ToUnicode` tabulka, ve které se nadefinuje `Unicode` symbol pro každý symbol z CID tabulky (někdy označované jako objekt `Differences` ve struktuře dokumentu).

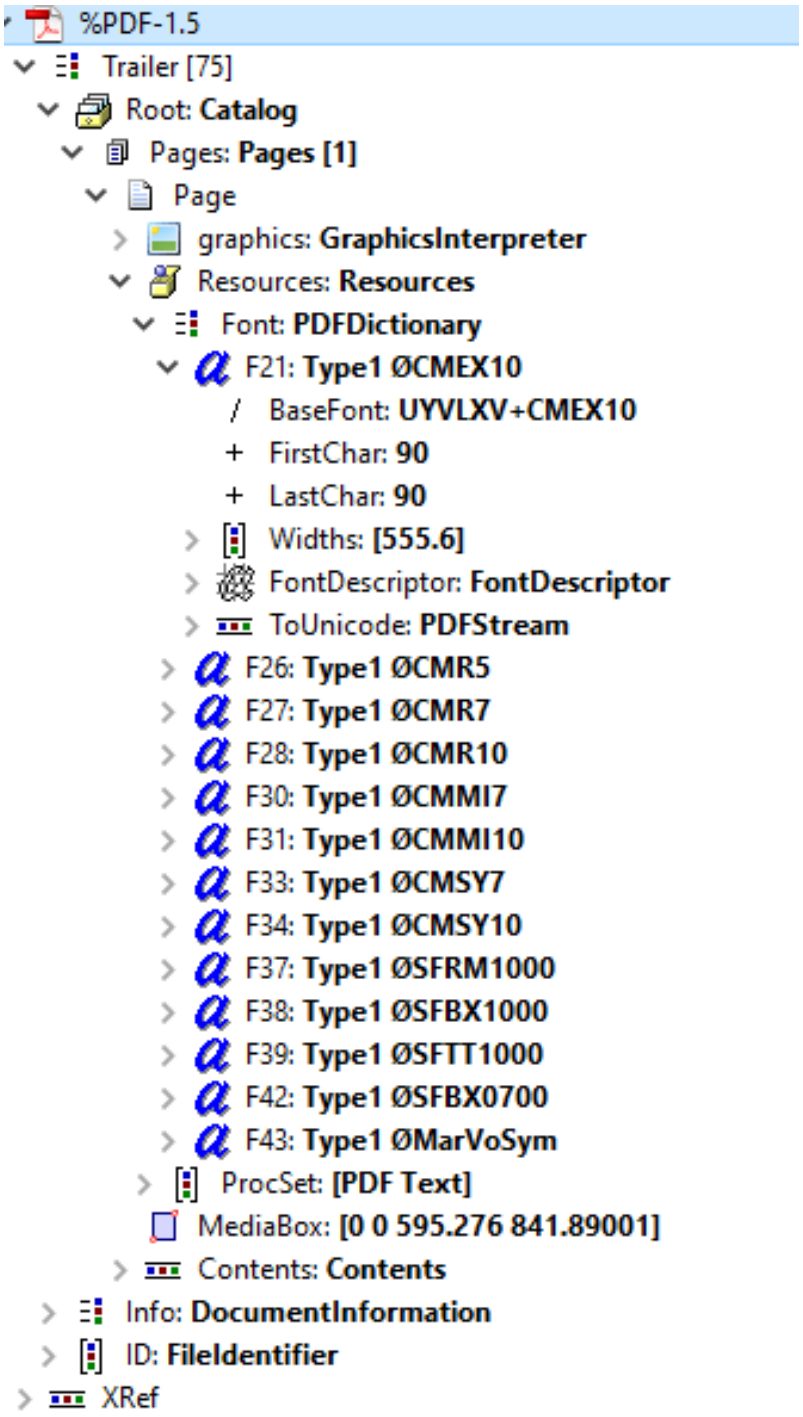
CID a `ToUnicode` tabulky lze zobrazit například pomocí programu `PDFtalk Snooper`, který zobrazí vnitřní strukturu celého dokumentu i s obsahy jednotlivých objektů. Na obr. 1.4 je zobrazena struktura jednostránkového PDF dokumentu vytvořeného pomocí LaTeXu, je v ní rozbalený objekt `Pages[1]`, což je objekt obsahující určité množství stránek – v tomto případě jednu – dále `Page` (první strana) → `Resources` → `Font` (ve kterém je seznam všech použitých fontů) → `F21`. Objekt `F21` obsahuje `ToUnicode` tabulku.

Na obr. 1.5 je zobrazen obsah objektu `ToUnicode`, který je typu `ByteString`. Každý řádek odpovídá mapování jednoho glyfu k jednomu `Unicode` symbolu. Za komentářem označeným symbolem „#“ je uveden symbol který odpovídá hexadecimální hodnotě uvedené ve druhém sloupečku. V prvním sloupečku jsou vypsány hodnoty CID. Ve stejném fontu je objekt `Differences` typu `Array`, jehož obsah vidíme na obr. 1.6, v prvním sloupečku je opět hodnota CID a ve druhém je označení glyfu odpovídající znaku ve fontu. Takto mapovaný font se bude správně zobrazovat, kopírovat a je možno v něm správně vyhledávat.

Na obr. 1.3 je vizualizováno toto kódování. Font obsahuje mapování glyfů a ID glyfů (GID), hodnoty CID a GID společně tvoří tabulku `Differences` a číselný `Unicode` ekvivalent a CID tvoří `ToUnicode` tabulku.



Obr. 1.3: Vizualizace kódování



Obr. 1.4: Struktura jednostránkového souboru vytvořeného pomocí LaTeXu

```
<2> <0046>  #F
<3> <0069>  #i
<4> <0072>  #r
<5> <006d>  #m
<6> <0061>  #a
<7> <0020>  #mezera
<8> <0043>  #C
<9> <006f>  #o
<a> <006e>  #n
<b> <0065>  #e
<c> <0063>  #c
<d> <0074>  #t
<e> <004f>  #O
<f> <0062>  #b
<10> <0079> #y
<11> <006c> #l
<12> <007a> #z
<13> <017e> #ž
```

Obr. 1.5: Obsah objektu `ToUnicode` uvnitř fontu

2	G70
3	G105
4	G114
5	G109
6	G97
7	G32
8	G67
9	G111
10	G110
11	G101
12	G99
13	G116
14	G79
15	G98
16	G121
17	G108
18	G122
19	G158

Obr. 1.6: Obsah objektu `Differences` uvnitř fontu

1.2.1 Standardy fontů

Standardy fontů jsou specifikace, které definují, jak mají být vytvářena, uložena a prezentována písma v digitálním prostředí. Tyto standardy zajišťují kompatibilitu mezi různými operačními systémy, textovými editory a dalšími aplikacemi, což umožňuje konzistentní zobrazení písma na různých zařízeních.

Na tabulce 1.1 je 8 standardů fontů podporovaných formátem PDF společně s jejich podporovanými typy kódování.

Tab. 1.1: Tabulka standardů typů fontů, převzato z [4]

Standardy typů fontů	**Podporované typy kódovacích standardů**
Type1	WinAnsi, MacRoman, MacExpert, Standard encoding a vestavěné kódování
CFF	WinAnsi, MacRoman, MacExpert, Standard encoding a vestavěné kódování
Multiple master (rozšíření Type1)	WinAnsi, MacRoman, MacExpert, Standard encoding a vestavěné kódování
TrueType	WinAnsi, MacRoman a vestavěné kódování
OpenType	WinAnsi, MacRoman a vestavěné kódování
Type3	vždy vestavěné kódování
Type 0 CIDFont (popisy glyfů na základě CFF)	Identity a CJK kódování
Type 2 CIDFont (popisy glyfů založeny na TrueType glyph technology)	Identity a CJK kódování

Type1

Type1 fonty, někdy zvané PostScript Type 1, protože byly uvedeny na trh společností Adobe systems společně s PostScriptem v roce 1984. V lednu roku 2023 přestaly některé Adobe aplikace podporovat fonty Type1 kvůli jejich zastaralosti. Společnost doporučila používat fonty OpenType [5]. Type1 fonty jsou založeny na vektorových datech místo na rastrových obrazcích, což umožňuje zobrazení písma ve velkých i malých velikostech bez zhoršení kvality nebo přílišného zvětšení velikosti souborů.

Type1 fonty nepovolují více než 256 glyfů v jednom fontu. Existuje však variace Type1 fontů zvaná CID fonty, které umožňují obejít toto omezení. Stará zařízení (před rokem 2003) někdy nedokážou se CID fonty pracovat správně [5].

Dříve byly tyto fonty velice populární, ale dnes jsou nahrazeny modernějšími fonty typu TrueType a OpenType.

CFF

CFF neboli Compact Font Format je kompaktnější formát vycházející z Type1, který přináší několik optimalizací a změn v designu. CFF disponuje efektivnějším kódováním písmen, což umožňuje minimalizovat velikost souborů. Stejně jako Type1 je CFF tvořeno vektorovými znaky, díky čemuž má písmo vysokou kvalitu. CFF fonty mohou obsahovat kódování znaků, což umožňuje podporu `Unicode` a zobrazení znaků z různých jazyků.

Multiple master

Multiple master byl opět formát fontu vyvinutý firmou Adobe systems. Tento formát představoval inovativní přístup k tvorbě písmen a jejich variací, jako písma s proměnlivou váhou, šířkou nebo sklonem. Díky tomuto formátu je možné vytvářet písma s různými tvary v jednom fontu, bez potřeby tvoření fontu pro každou variantu zvlášť.

Multiple master písma jsou tvořena pomocí tzv. os, kde každá osa představuje určitou proměnlivou vlastnost (např. osa váhy, osa šířky).

Při vytváření konkrétní varianty písma byl používán mechanismus interpolace, díky kterému mohl software vygenerovat nové písmo s hodnotami vlastností mezi dvěma nebo více definovanými variantami.

Multiple master přestalo být využívané ve většině projektů, i přesto že představoval zajímavý koncept.

TrueType

TrueType je vektorový formát písma vytvořený společností Apple společně s Microsoftem. Dodnes je velice populární a široce používaný v operačních systémech windows a macOS. TrueType podporuje `Unicode` [5]. Tyto fonty jsou definovány s použitím B-spline křivek, což je variace na beziérovy křivky využívané v PostScriptových fontech. TrueType fonty disponují velice sofistikovaným ovládáním hintingu, tedy techniky používané na vyhlazování hran znaků pro dosažení plynulejšího vzhledu zejména na obrazovkách s vyšším rozlišením. Některá TrueType písma mohou být implementována ve formátu OpenType.

OpenType

OpenType je formát fontu společně vytvořený společnostmi Adobe a Microsoft, poprvé představen v roce 1996 jako nástupce TrueType a Type1, jejichž vlastnosti kombinuje a přináší mnoho dalších vylepšení. Tento formát je dodnes široce používaný. OpenType může zahrnovat různé znakové sady, včetně `Unicode`. OpenType

fonty mohou zahrnovat široký rozsah glyfů, jako jsou ligatury (jako např. vizuální spojení písmen „fi“), swash znaky a další. Tato písma také podporují proměnlivé fonty, tedy různé váhy a sklony přímo v jednom fontu.

Type3

Type3 je formát písma, který byl zaveden v rámci PostScriptu. Většina specifikací týkající se Type1 fontů je platná i pro Type3 [6]. Type3 umožňuje rastrové i vektorové znaky, proto nepodporuje hinting. Tyto fonty mají vyšší flexibilitu znaků, ale taky těžší syntaxi kódování. V současné době se Type3 používá málo.

Type 0 CIDFont

Type 0 CIDFont (Composite Identity Font) je specifický typ fontu v rámci formátu PDF. Tento typ fontu je používán pro zobrazení znaků pro CJK (Čínština, Japonština, Korejština) jazyky a další písma, které vyžadují široký rozsah znaků. „Composite“ znamená, že font může obsahovat znaky z různých podfontů.

Type 0 CIDFont může používat kompresi pro efektivnější ukládání dat a snížení velikosti souboru PDF.

Každý znak je identifikován pomocí unikátního čísla CID. Tato identifikace umožňuje kódovat a reprezentovat velký počet znaků. Mapování CID na GID je klíčovým prvkem ve struktuře Type 0 CIDFont. Tato mapa definuje vztah mezi CID a GID.

Type 2 CIDFont

Type 2 CIDFont má vesměs stejné specifikace jako Type 0 CIDFont, liší se pouze v efektivnější kompresi dat.

1.3 Proč jsou některé dokumenty špatně kódovány?

V dnešní době je standardem, že každý font má `ToUnicode` tabulku. U některých dokumentů však tato tabulka chybí (např. protože byly vytvořeny v zastaralém SW).

Různé programy na zobrazování PDF používají jiné strategie na kopírování textu z dokumentů, kde není definována `ToUnicode` tabulka. Pokud prohlížeč nenalezne žádné řešení, pak program může text zkopírovat např. jako CID, které převede na `Unicode` ekvivalent. Pro příklad je uveden text, který byl zkopírovaný ze špatně kódovaného PDF dokumentu na obr. 1.7 a vložen do textového pole, kde se zobrazil jako text na obrázku 1.8. Text byl zkopírován do vyhledávacího pole prohlížeče

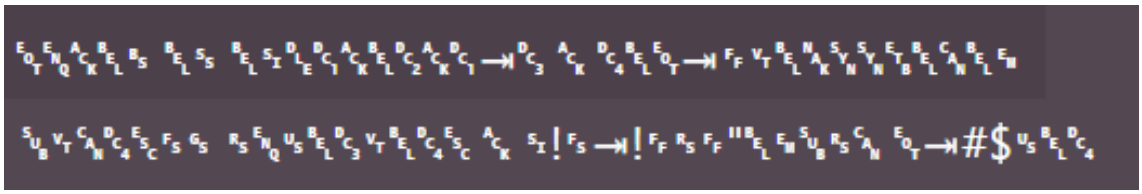
Google Chrome, který vizuálně zobrazil i řídící (netisknutelné) znaky, díky tomu lze s použitím tab. 1.2 vidět, že znaky opravdu měli, postupně přiřazeny CID, které byli, následně převedeny na `Unicode` znaky.

Z této ukázky je možné snadno pochopit, proč je v takto kódovaném dokumentu obtížné, často i nemožné, vyhledávat. Při pokusu k vyhledávání v takovém dokumentu např. slovo „byla“, program vám nenalezne žádné výsledky, nebo nalezne výsledky, ale v jiném než ve špatně kódovaném fontu nebo zobrazí místo, ve kterém se náhodou špatně namapovalo hledané slovo.

Všechna vydání časopisů AMARO využívají standard fontu Type1.

Firma Connect One byla založena
v roce 1996 s přesvědčením, že většina
budoucích přístrojů,

Obr. 1.7: Zkopírovaný text



Obr. 1.8: Vložený text zobrazený jako převod CID na `Unicode`

Tab. 1.2: Tabulka `Unicode` znaků viditelných na obr. 1.7

Unicode ekvivalent	Dec	Zkratka / glyf	Unicode ekvivalent	Dec	Zkratka / glyf
U+0004	4	EOT	U+0015	21	NAK
U+0005	5	ENQ	U+0016	22	SYN
U+0006	6	ACK	U+0017	23	ETB
U+0007	7	BEL	U+0018	24	CAN
U+0008	8	BS	U+0019	25	EM
U+0009	9	HT	U+001A	26	SUB
U+000A	10	LF	U+001B	27	ESC
U+000B	11	VT	U+001C	28	FS
U+000C	12	FF	U+001D	29	GS
U+000D	13	CR	U+001E	30	RS
U+000E	14	SO	U+001F	31	US
U+000F	15	SI	U+0020	32	(mezera)
U+0010	16	DLE	U+0021	33	!
U+0011	17	DC1	U+0022	34	"
U+0012	18	DC2	U+0023	35	#
U+0013	19	DC3	U+0024	36	$
U+0014	20	DC4			

2 Návrh a realizace řešení

K realizaci řešení je potřeba nejprve rozumět vnitřní struktuře PDF a jeho kódování.

Typ špatného kódování, který skript opraví je případ, kdy font má tabulku `Differences` s CID a glyfy, ale nemá `ToUnicode` tabulku. K sestavení této tabulky se skenuje každý řádek `Differences` tabulky a ke každému glyfu se přiřazuje `Unicode` ekvivalent podle námi vytvořené tabulky pro aktuální font. Problém této metody je, že některé znaky jako např. „^{2}“ může někdy být uložený pod CID, které reálně neodpovídá pozici zmíněného znaku. Případy, jako je tento, se obvykle ukládají do CID 2 až 32 pro glyfy „G0“ až „G30“ protože na těchto pozicích jsou mapovány řídící netisknutelné znaky, které ve většině moderních dokumentů nemají využití.

Důležité je si uvědomit, že toto je pouze jeden příklad špatného kódování, který platí pouze pro některé dokumenty a některé fonty. Konkrétně se zde opravují některé fonty Type1. V Časopisech AMARO mají různé ročníky jiné problémy s kódováním. Následující řešení funguje u vydání časopisů (z dodaných vzorků) „Praktická elektronika“ (PE) z roku 2008, PE 2010, PE 2013, PE 2015, PE 2016 a PE 2017, částečně PE 2011, PE 2012, PE 2014.

Časopisy AMARO obsahují rozdílné fonty pro různé variace písma, např. `Arial` tučný, `Arial` kurzíva, `Arial` v různých velikostech, což však není příliš problematické, protože i v těchto různých fontech jsou dodrženy stejné GID. Pro uživatele, který chce kopírovat text z dokumentu, je ve většině případů nežádoucí, aby bylo dodrženo původní formátování, proto nevadí, že se písmena napsaná kurzívou namapují a následně zkopírují jako vzpřímená písmena. Z tohoto důvodu je nutno v dalším rozvoji skriptu vytvořit tabulky pro různé nejčastěji používané fonty, které skript rozliší.

Další proměnná, kterou je třeba vzít v potaz, je že dokumenty obsahují různé fonty, kde každý font má jinak mapované GID, tedy pokud by se uživatel pokusil opravit dokument pomocí skriptu, který opraví pouze font `Arial` na dokument, který obsahuje font `Times`, mapování by opět bylo chybné.

V této kapitole bude popsán postup k vytvoření skriptu, který získá tabulku `Differences`, z každého glyfu získá `Unicode` ekvivalent, vytvoří tabulku `ToUnicode`, kterou následně vloží do původního PDF dokumentu, který uloží jako nový soubor. Tento proces následně opakuje pro každý PDF dokument v určené složce.

2.1 Filtrování fontů bez `ToUnicode` tabulky z PDF

Prvním krokem k opravě kódování bylo získat data z dokumentu pomocí `Pythonu` a knihovny `pikePDF`. K tomu byl vytvořen následující kód s pomocí dokumentace použité knihovny [7].

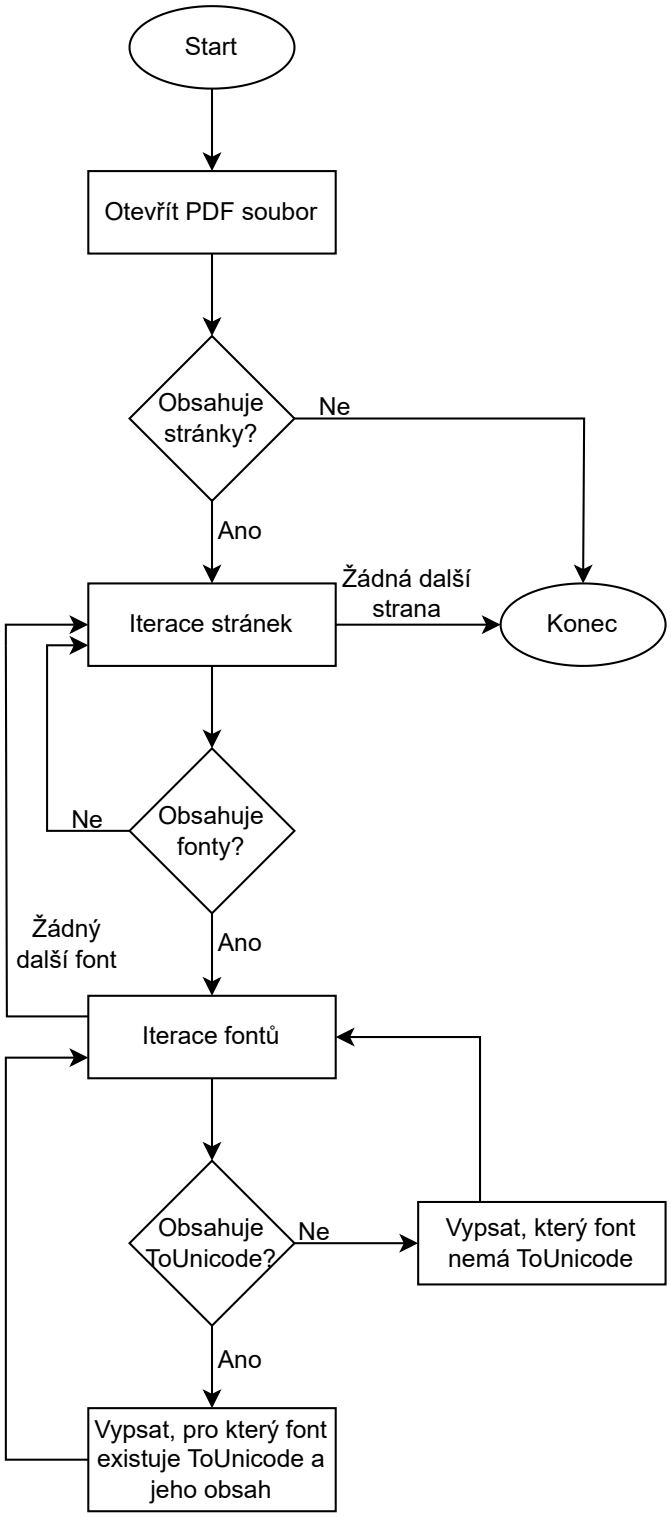
```
import pikepdf #importování knihovny pikepdf

#otevření souboru pomocí pikePDF
with pikepdf.open('input.pdf') as pdf:
  for page in pdf.pages:     #procházíme každou stranu
    #zjistíme zda strana obsahuje fonty
    if page.Resources.Font is not None:
      fonts = resources.Font
      for font_key, font_val in fonts.items():
        try:
          if font_val.ToUnicode is not None:
            toUni = font_val.ToUnicode
            #můžeme vypsat obsah ToUnicode tabulky pokud
                existuje
            print(toUni.read_bytes())
          if toUni is not None:
            #Oznámení v konzoli pro jakou stranu byla
               nalezena ToUnicode tabulka
            print(f'Found␣ToUnicode␣table␣for␣font␣{
               font_key},␣page␣{page.label}')
        except:
          #pokud došlo k chybě, pak se oznámí, že strana
              nemá ToUnicode tabulku
          print(f'No␣ToUnicode␣table␣for␣font␣{font_key},
             ␣page␣{page.label}')
```

Kód otevře soubor, se kterým bude pracovat a iteruje jeho stránkami, na každé stránce iteruje jejími fonty, v tomto bodě je možno určit zda má font tabulku `ToUnicode` a to tak, že nenastane chyba při pokusu o její přečtení. Pokud tabulka existuje vypíše se její obsah, pokud ne, pak se napíše zpráva, že font nemá `ToUnicode` tabulku.

Kód vypíše všechny fonty, pro které byla nalezena `ToUnicode` tabulka a vypíše jejich obsah.

Na obr. 2.1 je vývojový diagram výše uvedeného kódu.



Obr. 2.1: Vývojový diagram kódu pro získání `ToUnicode` tabulek

2.2 Úprava `ToUnicode` tabulky uvnitř PDF

K editaci `ToUnicode` tabulky lze použít předchozí kód až do bodu, kdy se kód dostane k fontu, který bude upravovat. Poté může v nové proměnné vytvořit tabulku, kterou přidá do souboru a vloží ji následujícím kódem.

```
    new_ToUni = b'<2> <0046>\n' \
                b'<3> <0069>\n' \
                b'<4> <0072>\n' \
                b'<5> <006d>\n' \
                b'<6> <0061>\n' \
                b'<7> <0020>\n' \
                b'<8> <0043>\n' \
                b'<9> <006f>\n' \
                b'<a> <006e>\n' \
                b'<b> <0065>\n' \
                b'<c> <0063>\n' \
                b'<d> <0074>\n' \
                b'<e> <004f>\n' \
                b'<f> <0062>\n' \
                b'<10> <0079>\n' \
                b'<11> <006c>\n' \
                b'<12> <007a>\n' \
                b'<13> <017e>\n'
    font_val.ToUnicode.write(b, filter=None, decode_parms
       =None, type_check=True)
pdf.save('output.pdf')
```

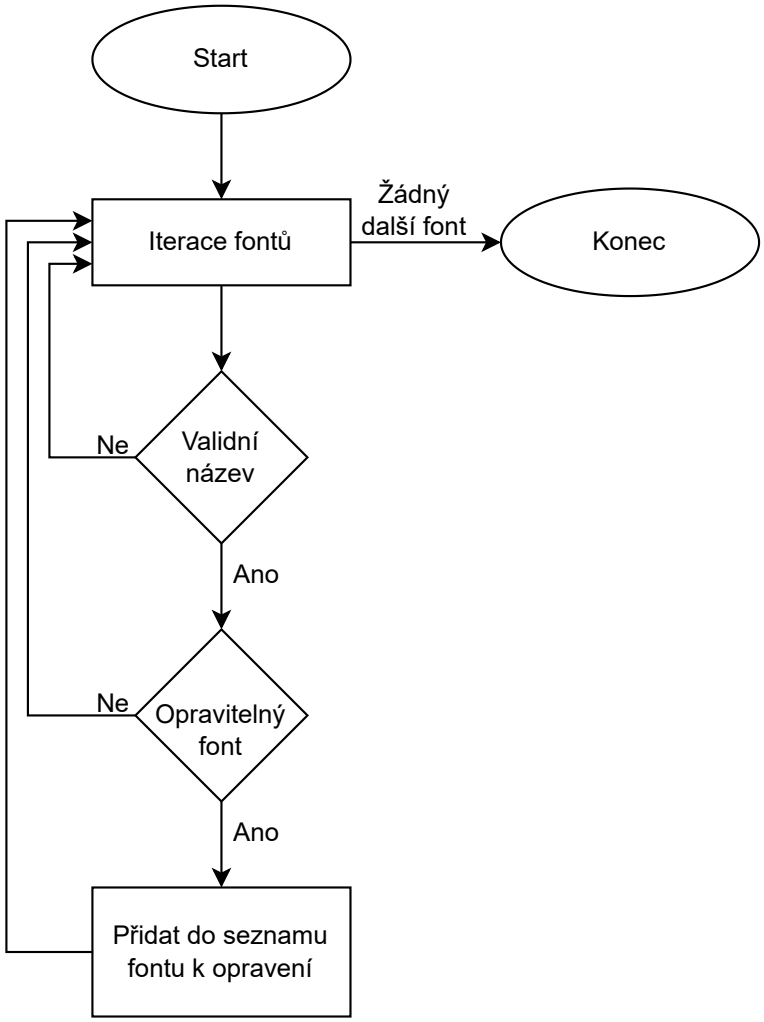
Po vložení nové tabulky je nutné uložit výsledek jako nový soubor. Tabulka vytvořená v předchozím případě je specifická pro jeden font a jednu stranu v jednom konkrétním souboru. Data v ní jsou založeny na tabulce `Differences`, která je na obrázku 1.6. Vytváření takových tabulek je dosti časově náročné, a proto došlo k automatizaci pomocí skriptu.

Správné vytvoření objektu `ToUnicode` tabulky je velice složité. Jedno z možných řešení je převzít tabulku z jiného PDF dokumentu, vložit ji do dokumentu, který je nutno opravit a přepsat její obsah.

2.3 Ověření zda je font opravitelný

```
for font_key, font_val in fonts.items():
    fontName = ''
    try:
        fontName = str(font_val.FontDescriptor.FontName)
    except:
        continue
    if "Arial" in fontName:
        try:
            if font_val.Encoding is not None and font_val
               .Encoding.Differences is not None:
                if font_key not in fixable_fonts:
                    fixable_fonts.append(font_key)
        except AttributeError:
            print("Differences attribute does not exist."
               )
```

První se v kódu ověří, zda současný font má jméno (atribut `font_val.FontDescriptor.FontName`), pokud ne pak se přejde rovnou na další font. Po nalezení názvu fontu se přejde k ověření, zda se jedná o opravitelný font (hledání klíčového slova v názvu fontu: (`"Arial"in fontName`). V další podmínce se ověřuje existence tabulky `Differences` a atrubutu `Encoding` ve fontu. Po splnění všech těchto podmínek se fontu může přiřadit vlaječka `fixable`, což se provede uložením označení fontu do seznamu opravitelných fontů `fixable_fonts`. Vývojový diagram tohoto kódu je na obr. 2.2. Důvod proč v tomto kódu označujeme pouze fonty `Arial` jako opravitelné je, že v rámci této práce byl vytvořen slovník pouze pro tento font. Po vytvoření slovníků pro další fonty je možné kód rozšířit i pro jejich opravu. V obdržených vzorcích je největší část textu ve fontu `Arial`, proto byl vybrán jako hlavní font k opravě.



Obr. 2.2: Vývojový diagram kódu pro získání seznamu opravitelných fontů

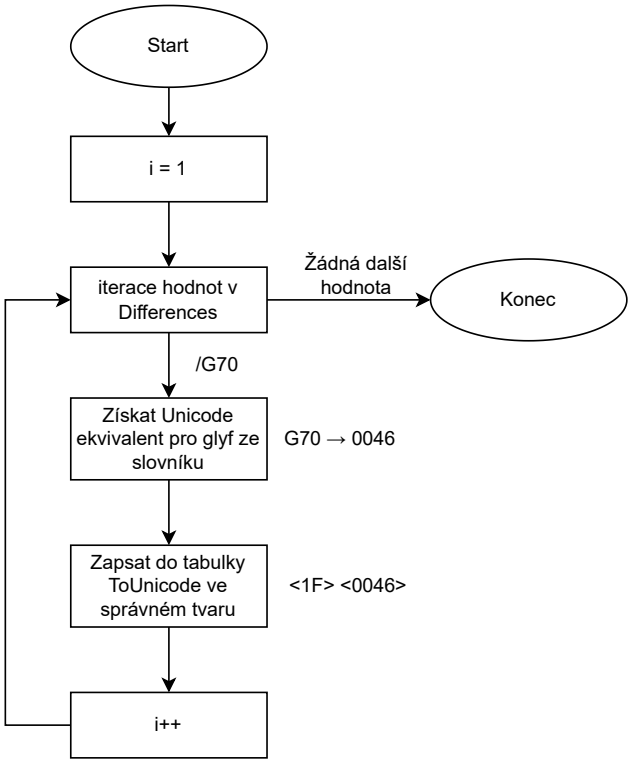
2.4 Vytvoření `ToUnicode` tabulky z `Differences`

Po identifikaci opravitelného fontu je potřeba vygenerovat tabulku `ToUnicode` pro následnou injekci do fontu.

```
i = 1
for val in font_val.Encoding.Differences:
    uni_from_dict = ArialDict.arial_dict.get(str(val)
       [1:])
    if i < 16:
```

```
        toUni_generated += f"<0{hex(i)[2:].upper()}> <{
            uni_from_dict}>\n".encode()
    else:
        toUni_generated += f"<{hex(i)[2:].upper()}> <{
            uni_from_dict}>\n".encode()
    i += 1
```

Kód iteruje hodnotami (`val`) v tabulce `Differences` (tedy atribut `font_val.Encoding.Differences`) získaná hodnota je glyf ID např. ve tvaru „/G70“. Jelikož ve slovníku jsou hodnoty pouze ve tvaru např. „G70“ je potřeba odebrat znak „/“ a získat hodnotu ze slovníku pomocí zbylého klíče, tím získáme hodnotu `uni_from_dict`, která je číselným označením Unicode ekvivalentu např. ve tvaru 0046. Následně se do generovaného `ToUnicode` objektu zapíše záznam o tomto znaku ve tvaru např. `<1F> <0046>`, tato data se zapíšou v binárním tvaru. První hodnota (tedy 1F) je CID, tedy pořadí znaků. CID je při zapisování převedeno na hexadecimální hodnotu. V kódu se tato hodnota získá z indexu `i`, který se inkrementuje při každé iteraci hodnoty z `Diffferences`. CID musí být dvouciferná hodnota, kde hexadecimální písmena musí být velkým písmem, toto je ošetřeno podmínkou v kódu. Ke správné funkčnosti PDF je nutné, aby délka `ToUnicode` byla rovna délce `Differences`, podmínka která je v kódu zaručena vytvořením řádku `ToUnicode` pro každý řádek `Differences`. Na obr. 2.3 je vývojový diagram předešlého kódu.



Obr. 2.3: Vývojový diagram kódu na vytvoření `ToUnicode` tabulky

2.5 Iterace a oprava všech zadaných souborů

Soubory, které chce uživatel opravit, musí nakopírovat do složky `input`, kde si je skript najde a pracuje s nimi. Pomocí knihovny `os` najde skript cestu k adresáři, kde se nachází a také cesty k adresářům `input` a `output`.

```
# Najít současný adresář
current_directory = os.path.dirname(os.path.abspath(
    __file__))
# Najít adresáře input a output
input_directory = os.path.join(current_directory, 'input'
    )
output_directory = os.path.join(current_directory, '
    output')
```

Ze souborů v adresáři `input` vytvoří seznam a z jednotlivých cest vyextrahuje názvy souborů, pomocí kterých se budou vygenerované soubory ukládat.

```
# Najít všechny .pdf soubory v adresáři input (včetně
   cesty)
pdf_files = glob.glob(os.path.join(input_directory, "*.
   pdf"))
# Extrahovat pouze názvy souborů
pdf_files = [os.path.basename(pdf) for pdf in pdf_files]
```

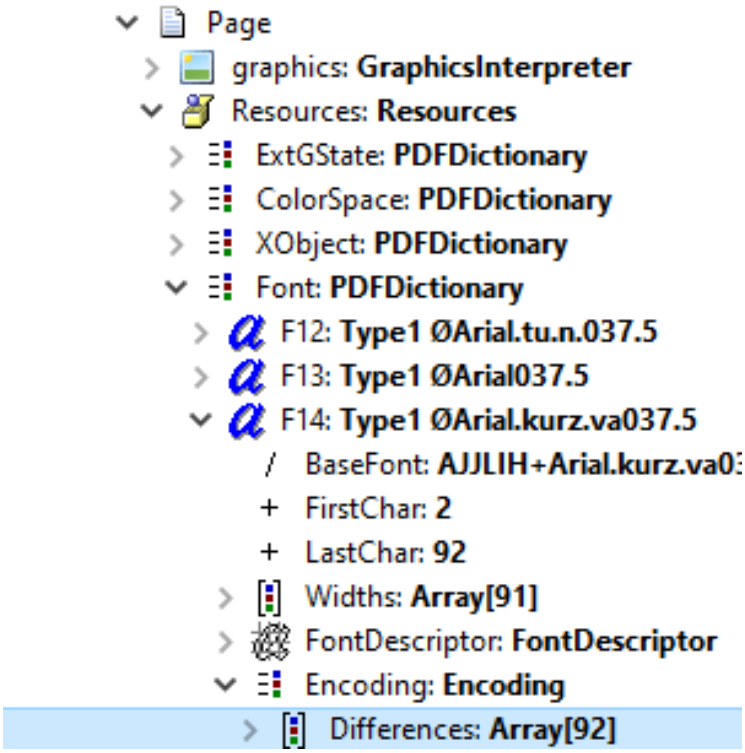
Vytvořeným seznamem skript iteruje a opravuje soubory v seznamu.

```
for file_name in pdf_files:
    # Získání cesty k souboru
    input_path = os.path.join(input_directory, file_name)
    # Vytvoření cesty a názvu nového souboru
    output_path = os.path.join(output_directory,
       file_name[:-4] + '_toUnicode.pdf')

    # Soubor se přeskočí, pokud uživatel vloží již
       opravený soubor do inputu
    if file_name.endswith("_toUnicode.pdf"):
        continue
```

2.6 Vytvoření překladového slovníku glyf na `Unicode`

K vytvoření `ToUnicode` tabulky je potřeba mít slovník překladu glyf ID na `Unicode` ekvivalent. Vytvoření tohoto slovníku je časově náročné a je pouze manuální práce. Naštěstí stačí vytvořit jeden slovník k opravě všech variací konkrétního fontu, postup k sestavení tohoto slovníku je následovný. Otevře se soubor s fontem, pro který je nutno vytvořit slovník v prohlížeči PDF a také v PDFtalkSnooper. V PDFtalkSnooper se nalezne font, který je třeba opravit a jeho `Differences` tabulka, tak jak je zobrazena na obr.2.4.



Obr. 2.4: Lokace `Differences` ve fontu

V tabulce `Differences` jsou glyf ID vypsány ve stejném pořadí, v jakém se používají v textu, díky tomu je možno určit, který znak patří ke kterému glyf ID. Po spárování znaků a glyf ID je nutno převést znaky na jejich `Unicode` evkivalenty, k takovému převodu lze použít tabulku `Unicode` znaků nebo online převodník. V tuto chvíli jsou dostupné všechny údaje potřebné k vytvoření slovníku, stačí je tedy vypsat do `Python` objektu typu `Dictionary` ve tvaru, který je znázorněn v následujícím kódu. V rámci práce byl vytvořen slovník pro font `Arial`.

```
arial_dict = {
    "G32": "0020",
    "G33": "0021",
    "G34": "0022",
    "G35": "0023",
    "G36": "0024",
    "G37": "0025",
    "G38": "0026",
}
```

2.7 Návod k instalaci a použití skriptu

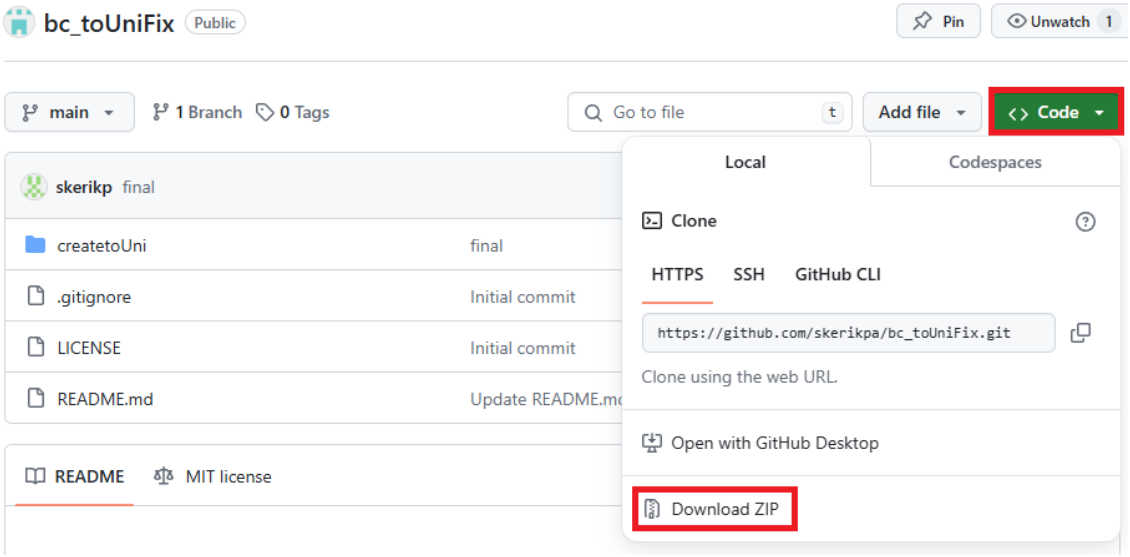
Ke spuštění skriptu je nutné stáhnout a nainstalovat `Python` (Dostupný ze stránek `www.python.org`), k vzpracování této práce byl použit `Python` verze 3.11. Po instalaci `Pythonu` je třeba nainstalovat balíčkový manažer `pip`, který se použije k instalaci knihovny `pikePDF`. Pro instalaci `pip` stačí napsat do příkazového řádku následující příkaz:

```
$ python get-pip.py
```

Při správné instalaci `pip` je možno stáhnout knihovnu `pikePDF` pomocí následujícího příkazu v příkazovém řádku:

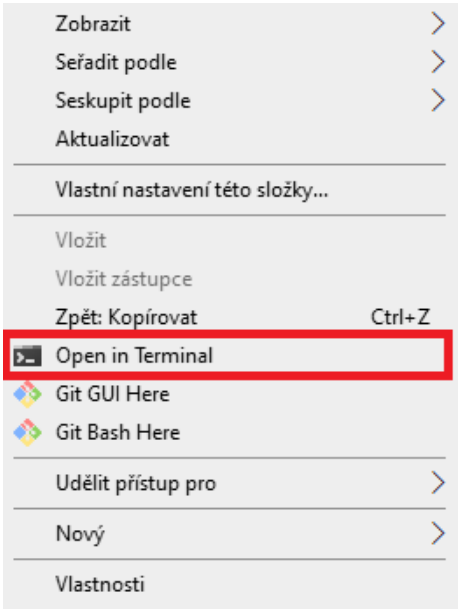
```
$ pip install pikepdf
```

V tomto bodě je nainstalováno vše, co je potřeba ke spuštění skriptu. Samotný skript je ke stažení na stránce `https://github.com/skerikpa/bc_toUniFix` pomocí tlačítka **Code** a následně **Download ZIP** dle obr.2.5, případně je možno stáhnout skript pomocí aplikace `GitHub Desktop` nebo naklonovat z příkazového řádku aplikace `Git Bash`. Při stažení archivu se soubor ZIP extrahuje v požadované lokaci.



Obr. 2.5: Tlačítko pro stažení skriptu

Po extrakci se otevře složka se skriptem, která také obsahuje složky `input` a `output`. Do složky input se vloží soubory potřebující opravu a do složky `output` vygeneruje skript opravené soubory. Skript se spustí pravým kliknutím do složky se skriptem a otevře se příkazový řádek tlačítkem **Open in terminal** (otevřít v konzoli) dle obr.2.6.



Obr. 2.6: Tlačítko ke spuštění konzole

Do konzole stačí už pouze napsat následující příkaz a skript se spustí.

```
$ python .\main.py
```

Dokončení práce skriptu lze poznat podle posledního řádku výpisu, na kterém je napsáno „Hotovo.“

Závěr

V rámci práce jsem se seznámil se strukturou PDF dokumentů a kódování jejich fontů. Cílem práce bylo vytvořit skript pro opravu kódování v PDF dokumentech pomocí jazyka `Python` a knihoven schopných upravovat vnitřní strukturu PDF.

Vytvořený skript umožňuje opravu všech fontů `Arial` pro všechny uživatelem zadané soubory, které mají platnou tabulku `Differences` a současně jim chybí tabulka `ToUnicode`. V sekci 2 byly vyjmenovány vzorky, které je možno opravit pomocí tohoto skriptu, tedy „Praktická elektronika“ (PE) z roku 2008 (12 vzorků), PE 2010 (12 vzorků), PE 2013 (13 vzorků), PE 2015 (12 vzorků), PE 2016 (13 vzorků) a PE 2017 (12 vzorků). Dohromady je skript schopný opravit 74 dodaných vzorků.

K opravě dalších fontů, nejen `Arial`, je potřeba vytvořit slovníky pro překlad glyf ID na `Unicode` ekvivalent. V rámci práce byla vytvořena pouze tabulka pro font `Arial`, protože nějvětší část textu v dodaných vzorcích je právě v tomto fontu.

Skript umí automaticky zpracovat větší množství souborů, které byly vloženy do složky `input`.

Protože různé programy k prohlížení PDF používají různé strategie pro práci s `ToUnicode` tabulkou a v současném stavu skript nedokáže vytvořit hlavičku této tabulky, je soubor opraven jen pro některé programy. Programy, které byly testovány a fungují správně jsou: MS Edge, Google Chrome, SumatraPDF v3.5.2 a Mozilla Firefox, naopak programy, ve kterých se znaky kopírují nesprávně i po opravě jsou: Adobe Acrobat Reader, SumatraPDF v3.2 a X-Change Viewer.

K vývoji skriptu byly použity pouze open-source softwary. Do skriptu byly přidány komentáře a návod k instalaci a použití byl přidán do `README.md` souboru na `GitHubu`. Výsledný skript je volně dostupný ke stažení na stránkách `https://github.com/skerikpa/bc_toUniFix`

Literatura

[1] ADOBE. Co znamenají písmena PDF?. Online. Dostupné z: https://www.adobe.com/cz/acrobat/about-adobe-pdf.html [citováno 2023-12-03]

[2] International organization for standardization. ISO 32000-1:2008. Online. Dostupné z: https://www.iso.org/standard/51502.html [citováno 2023-12-03]

[3] ADOBE. Evolution of the PostScript Language. Online. Dostupné z: https://www.adobe.com/products/postscript.html [citováno 2023-12-10]

[4] SANTHANAM, L. Font and Encoding Standard types supported in PDF for the representation of text content. Online. Dostupné z: https://www.gnostice.com/nl_article.asp?id=383&t=Font_and_Encoding_Standard_types_supported_in_PDF_for_the_representation_of_text_content [citováno 2023-12-11]

[5] LAURENS, L. Fonts in PDF files. Online. Dostupné z: https://www.prepressure.com/pdf/basics/fonts [citováno 2023-12-12]

[6] LAURENS, L. Type 3 fonts. Online. Dostupné z: https://www.prepressure.com/fonts/basics/type3 [citováno 2023-12-12]

[7] BARLOW, J. pikepdf Documentation. Online. Dostupné z: https://pikepdf.readthedocs.io/en/latest/tutorial.html [citováno 2023-12-04]

Seznam symbolů a zkratek

PDF	Portable Document File
ISO	International orgranization for standardization
CID	Character identification
GID	Glyph ID
CFF	Compact font format

A Přiložená ukázka opraveného souboru

V příloze se nacházejí soubory **__PE01__2010__ukazka.pdf** a **__PE01__2010__opravena__ukazka.pdf**. První zmíněný soubor je původní jednostránkový vzorek časopisu a druhý soubor je stejný soubor, který byl opraven skriptem vytvořeným v rámci této bakalářské práce.