**开源软件基础大作业**

**python实现Markdown语法的可视化演示工具**

**A Visual Presentation Tool for Markdown Syntax In Python**

学 生 姓 名： 张佳音（组长）

学 号： 201892397

学 生 姓 名： 沈佳眉

学 号： 201892069

学 生 姓 名： 陈昱桥

学 号： 201892203

完 成 日 期： 2021年1月10日

大连理工大学

目 录

[引 言 1](#_Toc61209941)

[1. 项目背景及意义 2](#_Toc61209942)

[2. 项目创新点 2](#_Toc61209943)

[3. 项目分析及设计 2](#_Toc61209944)

[3.1. 程序分析 2](#_Toc61209945)

[3.2. 代码设计 3](#_Toc61209946)

[3.2.1. pypt.py 3](#_Toc61209947)

[3.2.2. config.py 3](#_Toc61209948)

[3.2.3. window.py 3](#_Toc61209949)

[3.2.4. document.py/page.py 3](#_Toc61209950)

[4. 项目实现 4](#_Toc61209951)

[4.1. 代码实现 4](#_Toc61209952)

[4.1.1. pypt.py文件 4](#_Toc61209953)

[4.1.2. config.py文件 4](#_Toc61209954)

[4.1.3. document.py文件 5](#_Toc61209955)

[4.1.4. window.py文件 8](#_Toc61209956)

[4.2. 步骤及截图 9](#_Toc61209957)

[5. 项目测试 11](#_Toc61209958)

[6. 不足及可改进之处 13](#_Toc61209959)

[7. 总结 13](#_Toc61209960)

[8. 感想 14](#_Toc61209961)

[9. 成员分工 14](#_Toc61209962)

# 引 言

Python由荷兰数学和计算机科学研究学会的Guido van Rossum 于1990 年代初设计，提供了高效的高级数据结构，还能简单有效地面向对象编程。

Python是一种跨平台的计算机程序设计语言。 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，逐渐被用于独立的、大型项目的开发。

基于其简洁性、易读性和可扩展性，我们选择基于python实现markdown语法的可视化演示工具。

Markdown是为网络书写者而生的方便“text-to-HTML”工具，它允许人们用一种易读易写的纯文本格式进行书写。Markdown不用被纷繁的排版和样式打乱书写的节奏，把内容和结构从文档中分离出来，只需要关注实际的写作，而不用担心章节标题的样式怎么调整，段落间插入多少空白之类的问题。为用户提供了便捷和效率，大大提高了用户体验感。

同时，由于Markdown文档是由源码书写的纯文本文档，且内容与形式相分离，我们希望编写程序将在Markdown格式基础上微调的文件进行一个章节划分和展示。将document的成员划分出多级标题及内容。

# 项目背景及意义

传统的ppt文件采用富文本格式，文件结构复杂、体量大，并且需要特定的软件来编辑和展示，而相关软件本身也大量占用硬盘和内存；而且在编辑过程中，由于富文本格式大量考虑的是版式和印象而非内容本身，使得大量时间浪费在设计上，有时用户花费大量的时间和精力也未必能达到一个很好的效果，在展示过程中也过多关注设计而非内容。此外由于软件版本的差异，同样的文件展示的结果也有所差异，未更新版本的兼容性甚至会影响整个文件的效果。

与此同时，我们看到了markdown的优势，并决定使用兼容markdown语法的文档格式来作为我们的文档结构。在一定程度上允许用户通过编辑配置文件修改主题字体本色等，而所有排版相关的任务由程序完成，使得用户可以专注于编辑文档的内容而不必分心。

# 项目创新点

随着科技的进步，我们越来越多的通过打字的方式进行输入。电子版的表格、文档具有传输方便、易于修改等特点。网络小说、手机游戏的兴起更是带动了网络书写的普及和通用性。在特定的演讲、会议场合更需要排版精良的幻灯片来吸引眼球。

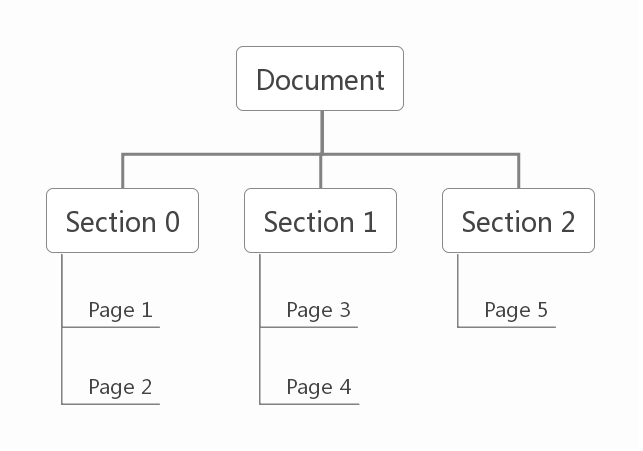
然而键盘录入并不能像手写一样，根据自身即时所想决定落笔位置、文字大小以及字间距离等问题。只能在键入前或后进行统一调整，文字量过大时便会带来许多麻烦。编辑、整理文件所用的大量时间和精力正是当代工作生活中的巨大痛点。

# 项目分析及设计

## 程序分析

关于document结构，它读取的是markdown格式的文件，因此拥有head1, head2, head3, list, bold/italic等元素，分页将按照标题和容量来划分。

document 下包含一个section类用于记录一个个章节；page下的 paragraph 类记录一个段落；paragraph类包括“内容”和“属性”，即该段落是普通段落还是标题段还是代码段。



一级标题默认和日期、作者展示在第一页；

二级标题为横向展开的分页，二级标题下的内容纵向排列；

三级标题则是作为内容纵向排列。

## 代码设计

### pypt.py

pypt.py是主文件，负责启动程序、初始化数据和建立窗口。

### config.py

config.py是配置管理类，负责接受命令行参数和读取配置文件数据。

使用configparser，由py内置支持提供了读写和管理配置信息的接口；getopt提供处理命令参数的接口；通过sys来获取Python解释器的信息。

### window.py

window.py是窗体类，规范窗口的组件和内容，负责数据的展示和刷新。

tkinter由python内置支持的图形接口，通过与Tk交互实现图形窗体和可视化操作，弹窗库 tkinter.messagebox；

pypandoc实现文件格式转换。

### document.py/page.py

document.py和page.py负责读取 md文件，其中 doc是建立文档的逻辑结构，pg存储每一个展示页的内容。

均运用logging库，在程序执行过程中记录输出了大量日志信息 便于程序功能的调试和分析；另通过日志分级功能 将日志信息分info/warning/error三类，以便于在release版本发布时屏蔽不必要的日志输出。

# 项目实现

## 代码实现

### pypt.py文件

#启动程序、初始化数据和建立窗口等

def main():

# Initialize configurations

conf = config.Config()

conf.readArgu()

conf.readConf()

conf.writeConf()

theme = config.Config.theme

doc = document.Document()

doc.readFile(config.Config.inName)

# Create window

win = window.Window(doc, theme)

# Allow resize

win.master.resizable(width=True, height=True)

# Begin main events loop

log.info('Begin main loop')

win.mainloop()

### config.py文件

#实现接受命令行参数

def readArgu(self):

# Command arguments

opts, \_ = getopt.getopt(sys.argv[1:],

'hi:c:', ['help', 'input=', 'config='])

for key, value in opts:

if key in ('-h', '--help'):

self.usage()

sys.exit()

elif key in ('-i', '--input'):

Config.inName = value

elif key in ('-c', '--config'):

Config.configName = value

程序运行时，通过 ./pypt.py -h或 ./pypt.py --help，查看帮助信息

./pypt.py -i file或 ./pypt.py --input=file，读取一个待展示的文件

./pypt.py -c xxx.conf或 ./pypt.py --config=xxx.conf，读取配置项

#读写配置文件数据

def readConf(self):

# Read configurations

log.info('Reading config file...')

Config.confParser.read(Config.configName)

for sec in Config.confParser.sections():

for key, value in Config.confParser.items(sec):

if 'font' == key:

Config.theme[sec][0] = value

elif 'size' == key:

Config.theme[sec][1] = value

elif 'form' == key:

Config.theme[sec][2] = value

log.info('Configurations loaded')

def writeConf(self):

# Write configurations

log.info('Generating config file...')

for key in Config.theme.keys():

if not Config.confParser.has\_section(key):

Config.confParser.add\_section(key)

Config.confParser.set(key, 'font', Config.theme[key][0])

Config.confParser.set(key, 'size', str(Config.theme[key][1]))

Config.confParser.set(key, 'form', Config.theme[key][2])

Config.confParser.write(open(Config.configName, 'w'))

log.info('Config file saved to ' + Config.configName)

### document.py文件

#文件中包含document类及section类，document类存放文件信息，section类存放section结构，记录章节。

class Section:

def \_\_init\_\_(self):

self.pages = []

class Document:

def \_\_init\_\_(self):

self.sections = []

self.title = ''

self.author = ''

self.date = ''

#在Document类中，用代码实现向文件中添加section和page

def addPage(self, pg):

logging.info('Adding new page')

self.sections[-1].pages.append(pg)

def addSec(self):

logging.info('Adding new section')

self.sections.append(Section())

#在Document类中，用代码实现向页面中添加paragraph。

paragraph 类记录一个段落，包括“内容”和“属性”，即该段落是普通段落还是标题段还是代码段。

def addPara(self, paraStr, pg):

if paraStr.startswith('# '):

paraStr = paraStr[2:]

logging.info('Found top title: ' + paraStr)

# Multiple top title

if '' != self.title:

error('Only one top title is allowed!')

self.title = paraStr

para = page.Paragraph(lv='h1', cont=paraStr)

elif paraStr.startswith('## '):

paraStr = paraStr[3:]

logging.info('Found sub title: ' + paraStr)

para = page.Paragraph(lv='h2', cont=paraStr)

elif paraStr.startswith('### '):

paraStr = paraStr[4:]

logging.info('Found third title: ' + paraStr)

para = page.Paragraph(lv='h3', cont=paraStr)

elif paraStr.startswith('- '):

logging.info('Found an unordered list')

para = page.Paragraph(lv='ul', cont=paraStr)

elif paraStr.startswith('1. '):

logging.info('Found an ordered list')

para = page.Paragraph(lv='ol', cont=paraStr)

elif paraStr.startswith('```'):

paraStr = paraStr.split('\n', 1)[1]

paraStr = paraStr.replace('\n```', '')

logging.info('Found a code block')

para = page.Paragraph(lv='cb', cont=paraStr)

else:

logging.info('Found a normal paragraph')

para = page.Paragraph(lv='p',

cont=paraStr.replace('\n', ' '))

# Lack top title

if '' == self.title:

error('A top title is needed before any!')

pg.paras.append(para)

#在Document类中，实现读取文件方法

def readFile(self, fileName):

# Read data from file

with open(fileName, 'r') as f:

lines = f.readlines()

paraStr = ''

pg = page.Page()

cbCtr = 0

for line in lines:

# Para break

line = line.strip()

if '' == line and cbCtr == 0:

if '' != paraStr:

self.addPara(paraStr, pg)

paraStr = ''

# Page break <- Manully

elif line.startswith('---') and cbCtr == 0:

logging.info('Page break')

...

pg = page.Page()

# paragraph break affected by code block

elif line.startswith('```'):

if cbCtr == 0:

cbCtr = cbCtr + 1

logging.info('Paragraph break')

...

# Page breakr<- Automatically

elif line.startswith('# ') and cbCtr == 0 or line.startswith('## ') and cbCtr == 0:

...

# Author

elif line.startswith(r'%author: '):

line = line[9:]

logging.info('Found author: ' + line)

if self.author == '':

self.author = line

else:

error("only one author is allowed!")

# Date

elif line.startswith(r'%date: '):

line = line[7:]

...

else:

if '' != paraStr:

paraStr += '\n'

paraStr += line

if 0 != len(pg.paras):

self.addPage(pg)

### window.py文件

文件中定义窗口类class Window(tkinter.Frame)继承Frame

#实现了构造窗体、刷新页面、规范窗口组件及内容等功能。

def \_\_init\_\_(self, doc, theme, master=None):

...

#创新实现了window 类中将 Vim 的 h/j/k/l 和方向键绑定为导航键，即翻页键

bindedKeys = ['<Escape>', '<q>','<h>', '<j>', '<k>', '<l>',

'<Left>', '<Down>', '<Up>', '<Right>', '<s>','o']

for seq in bindedKeys:

self.master.bind(sequence=seq, func=self.eventHandler)

log.info('Key bindings applied')

#window类的标题和页面具体内容用Text组件

def initWidgets(self):

...

# Top label

self.labelTop = tk.Label(self,

text=self.doc.title + ' - ' + self.doc.date,

font=self.theme['p'],

fg='white', bg='#252525',

height=1)

self.labelTop.pack(side='top', expand='no', fill='both')

# Bottom label

self.textBottom = tk.StringVar()

self.textBottom.set(self.doc.author + ' - <0,0>')

self.labelBottom = tk.Label(self,

textvariable=self.textBottom,

font=self.theme['p'],

fg='white', bg='#252525',

height=1)

self.labelBottom.pack(side='bottom', expand='no', fill='both')

log.info('Topbar & bottombar created')

# Content text

self.textField = tk.Text(self,

fg='white', bg='#252525', bd=0, wrap='none', yscrollcommand=self.scrollbar.set,

autoseparators=False)

self.textField.pack(expand='yes')

self.scrollbar.config(command=self.textField.yview)

self.textField.config(state='disabled')

log.info('Text field created')

...

## 步骤及截图

demo.md

|  |
| --- |
|  |
|  |
|  |
|  |

# 项目测试

Equivalence class:

1. No top title(N)
2. one top title with no paragraph before(Y)
3. one top title with paragraphs before(N)
4. more than one top titles(N)
5. No sub title(Y)
6. more that zero sub titles(Y)
7. No third title(Y)
8. More than zero third titles(Y)
9. No unordered list(Y)
10. More than zero unordered list(Y)
11. No ordered list(Y)
12. More than zero ordered list(Y)
13. No code block(Y)
14. More than zero code block(Y)
15. No paragraph(Y)
16. more than zero paragraph(Y)
17. No author(Y)
18. one author(Y)
19. more than one author(N)
20. No data(Y)
21. one date(Y)
22. more than one date(N)
23. no empty line before subtitle(Y)
24. empty lines before subtitle(Y)
25. no empty line before code block(Y)
26. empty lines before code block(Y)
27. every section break affected by encountering top title(Y)
28. every section break affected by encountering sub titles(Y)
29. every page break affected by encountering sub titles(Y)
30. every page break affected by encountering '---\n'(Y)
31. every page break affected by encountering the end of file(Y)
32. every paragraph break affected by encountering an empty line(Y)
33. every paragraph break affected by encountering a code block(Y)
34. every paragraph break affected by encountering '---\n'(Y)
35. every paragraph break affected by encountering sub titles(Y)
36. words not wrapped by`` appear normal(Y)
37. words wrapped by `` appearing shadow(Y)
38. words not wrapped by \*\* appear normal(Y)
39. words wrapped by \*\* appearing italic(Y)
40. words not wrapped by \*\* \*\* appear normal(Y)
41. words wrapped by \*\* \*\* appearing bolder(Y)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 1 | 2 | 3 | 4 | 5 |
| 覆盖的等价类 | E2,E5,E8,E9,E12,E13,E16,E17,E20,E27,E30,E32,E36,E39,E41 | E2,E6,E7,E10,E11,E14,E15,E18,E21,E25,E23,E28,E31,E38,E40 | E2,E6,E8,E12,E13,E14,E22,E24,E26,E33,E34,E37,E39,E40 | E1,E6,E7,E10,E11,E14,E15,E18,E21,E25,E23,E28,E31,E38,E40 | E3,E6,E7,E10,E11,E14,E15,E18,E21,E25,E23,E28,E31,E38,E40 |
| 预期结果 | ok | ok | ok | fail | fail |

# 不足及可改进之处

现有程序只是简单的将输入文本进行筛选分类，可能会存在细化不够的问题，而现有程序并没有提出问题或反馈的途径入口等。后期应添加更多收集用户需求的界面，从而达到更好的交互目的。

前期为了减少开发时间，界面较为简洁，也同时满足了高效率的需求。但主题较为单一，界面的美感设计都是后期需要考虑到的问题。我们可以通过添加不同窗口是否显示的开关选项，和明亮、深色模式等切换，丰富用户的视觉体验。

# 总结

在本项目中，我们小组使用兼容markdown语法的文档格式文档结构，实现了在 md 格式基础上微调的文件进行一个章节划分和展示。在这个过程中，我们不断测试和完善程序，修改代码段中的问题，例如如果代码段中有空行怎么办，如果代码段太长，一页可能显示不下；我们通过完善，易于上下切换页面、前后跳转章节；添加了功能，按s键可以把md文件保存到PDF，但暂不支持中文字符；添加滚动条，当页面在一个框内显示不下时，可以手动滚动显示。优化代码，关于window类的标题和页面具体内容后期改用Text组件而不是Label和Canvas。最终，所需功能基本实现。

# 感想

张佳音：python语言的包罗万象。 python能调用各种库使冗杂的代码精简，从而节省大量的时间。相比于C语言，python能用更短的代码实现更强大的功能。同时，对于开源的需求，GitHub平台很好的满足了这样的交流需求，一个小团队分工协作，解决问题，帮助同学掌握知识、运用知识，也是很有意义的收获。

沈佳眉：Python 是一门简洁，开源的高级编程语言。Python具有语法简单的特点。Python具有丰富的库的支持。如tkinter等等，都是非常优秀的库。很多东西都不需要自己动手重新创造，直接调用第三方的性能强大的库即可。通过这次作业的完成，我对Python有了更深刻的理解，能够更加熟练的运用这门语言，更加理解到了编程的魅力，也我明白了认真做事的重要性。

陈昱桥：我是开始接触python，也是第一次参与一个python的小项目。在开始代码的编写和测试对我来说是一个很大的挑战。因为队友的帮助和同学指导，我觉得对python语言及github这类工具的认识加深了，有些思想与C++区别还是较大，有时也会觉得麻烦，但主要的也了解到了python的简洁。经过这一次学习学到了新的知识并加以应用，觉得有很大的收获。

# 成员分工

张佳音：主要负责document.py文件，建立文档的逻辑结构，实现文档的读取，在page.py中实现document重构。

沈佳眉：主要负责window.py中类的实现，完善了窗口的设计；完成pypt.py，并完善整体程序代码。

陈昱桥：负责config.py中配置管理，对代码进行测试完善，并整理文档。