

目录

- [1 开始之前](#)
- [2 VSCode的基本操作](#)
 - [2.1 VSCode是什么的?](#)
 - [2.2 为什么选择vscode?](#)
 - [2.3 记事本的基本操作](#)
 - [2.4 VSCode的基本操作](#)
 - [2.4.1 新建文件](#)
 - [2.4.2 打开文件](#)
 - [2.4.3 打开文件夹](#)
 - [2.4.4 快捷键](#)
 - [2.4.5 ToDoing](#)
 - [2.5 授人与鱼不如授人与渔](#)
- [3 Python的基本知识](#)
 - [3.1 关于编程的基本知识](#)
 - [3.1.1 什么是程序?](#)
 - [3.1.2 程序是干什么的?](#)
 - [3.1.2.1 输出从0到n](#)
 - [3.1.2.2 记录点击次数的按钮](#)
 - [3.1.3 无论什么编程语言,你都需要会他们](#)
 - [3.1.3.1 注释](#)
 - [3.1.3.2 函数](#)
 - [3.1.3.3 运算符](#)
 - [3.1.3.3.1 根据功能对运算符分类](#)
 - [3.1.3.3.2 运算符的优先级问题](#)
 - [3.1.3.3.3 根据操作数对运算符进行分类](#)
 - [3.1.3.3.4 概念由人定义](#)
 - [3.1.3.3.5 运算符参与化简代码](#)
 - [3.1.3.4 数据类型](#)
 - [3.1.3.5 语句](#)
 - [3.1.3.5.1 普通语句](#)
 - [3.1.3.5.2 条件语句](#)
 - [3.1.3.5.3 循环语句](#)
 - [3.1.3.5.4 跳转语句](#)
 - [3.2 Python的基础语法](#)
 - [3.2.1 关于Python不得不说的事](#)
 - [3.2.1.1 打开Python idle](#)
 - [3.2.1.2 Python的语言特性](#)
 - [3.2.1.2.1 什么是解释型?](#)

- [3.2.1.2.2 什么是交互型?](#)
- [3.2.2 Python与众不同的缩进](#)
- [3.2.3 Python注释](#)
- [3.2.4 函数](#)
 - [3.2.4.1 函数的定义](#)
 - [3.2.4.2 函数的调用](#)
- [3.2.5 运算符](#)
- [3.2.6 数据类型](#)
 - [3.2.6.1 如何知道一个数数据的数据类型?](#)
 - [3.2.6.2 关于数据类型,我们应该关注什么?](#)
- [3.2.7 语句](#)
 - [3.2.7.1 条件语句](#)
 - [3.2.7.1 循环语句](#)
- [3.3 写完程序该做什么?](#)
 - [3.3.1 安装 pyinstaller 模块](#)
 - [3.3.2 开始第一次打包吧](#)
 - [3.3.2.1 最简单的打包方法](#)
 - [3.3.2.2 更改参数来控制打包](#)
 - [3.3.2.3 优化打包命令和使用spec文件](#)
 - [3.3.2.4 所有代码提供](#)
 - [3.3.3 分享与使用](#)
- [3.4 python不那么基础的基础](#)
 - [3.4.1 pip详细介绍](#)
 - [3.4.1.1 常用二级命令](#)
 - [3.4.1.2 更改镜像源](#)
 - [3.4.2 文件的处理](#)
 - [3.4.2.1 输入和输出](#)
 - [3.4.2.2 打开关闭文件](#)
 - [3.4.2.3 读取写入文件](#)
 - [3.4.2.4 光标位置相关函数](#)
 - [3.4.2.5 字符串处理部分](#)
 - [3.4.3 常用的三个模块](#)
 - [3.4.3.1 time模块](#)
 - [3.4.3.2 random模块](#)
 - [3.4.3.2.1 常用函数](#)
 - [3.4.3.2.2 特定范围的随机数生成](#)
 - [3.4.3.3 math模块](#)
 - [3.4.3.3 我比较喜欢的模块](#)
- [3.5 实战 解析上文中的代码](#)
 - [3.5.1 输出0到n](#)
 - [3.5.2 记录点击次数的按钮](#)
 - [3.5.2.1 分析和拆解需求](#)
 - [3.5.2.2 写核心代码](#)
 - [3.5.2.3 还需要改进的地方](#)

- [3.5.2.4 原代码提供](#)
- [3.6 星辰大海 未来你可能会遇到什么](#)
 - [3.6.1 语法部分](#)
 - [3.6.2 面向对象技术](#)
 - [3.6.3 什么是编程?](#)
 - [3.6.4 如何学习编程?](#)
- [3.7 自学python我想给你点什么](#)
 - [3.7.1 好的教学网站](#)
 - [3.7.2 书籍推荐](#)
 - [3.7.3 视频教程](#)
 - [3.7.4 善用AI](#)
- [4 Windows终端](#)
 - [4.1 CMD和Powershell](#)
 - [4.2 CMD窗口的理解和配置](#)
 - [4.2.1 联系资源管理器和cmd窗口](#)
 - [4.2.2 终端的配置](#)
 - [4.2.2.1 简单的美化](#)
 - [4.2.2.2 快捷键](#)
 - [4.3 常用命令](#)
 - [4.4 方便的小技巧](#)
 - [4.4.1 不想使用命令行了,好麻烦](#)
 - [4.4.2 在psh中使用cmd](#)
 - [4.4.3 帮助界面](#)
 - [4.4.4 一些约定俗成的小习惯](#)
 - [4.4.5 环境变量](#)
 - [4.5 Markdown文档](#)
 - [4.6 如何自学终端的命令?](#)
- [5 结束之后](#)

1 开始之前

我将会通过这个文件告诉你什么?

1. vscode的基本操作
2. python的基本知识
3. 终端的一些知识和一些简单的命令
4. 一些其他的电脑常识

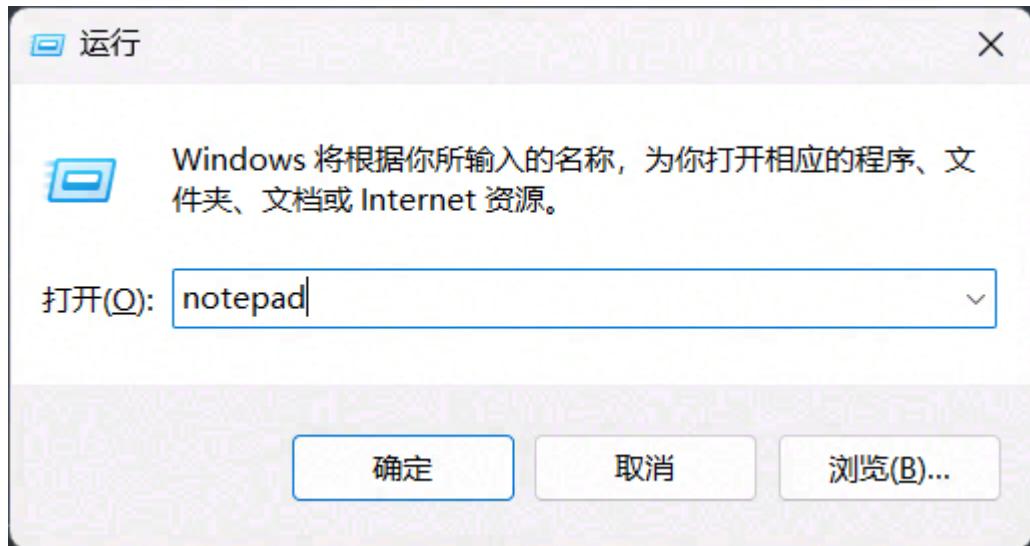
本文默认使用的是windows系统哦,使用linux或者macos的用户在部分内容上可能并不适用.

2 VSCode的基本操作

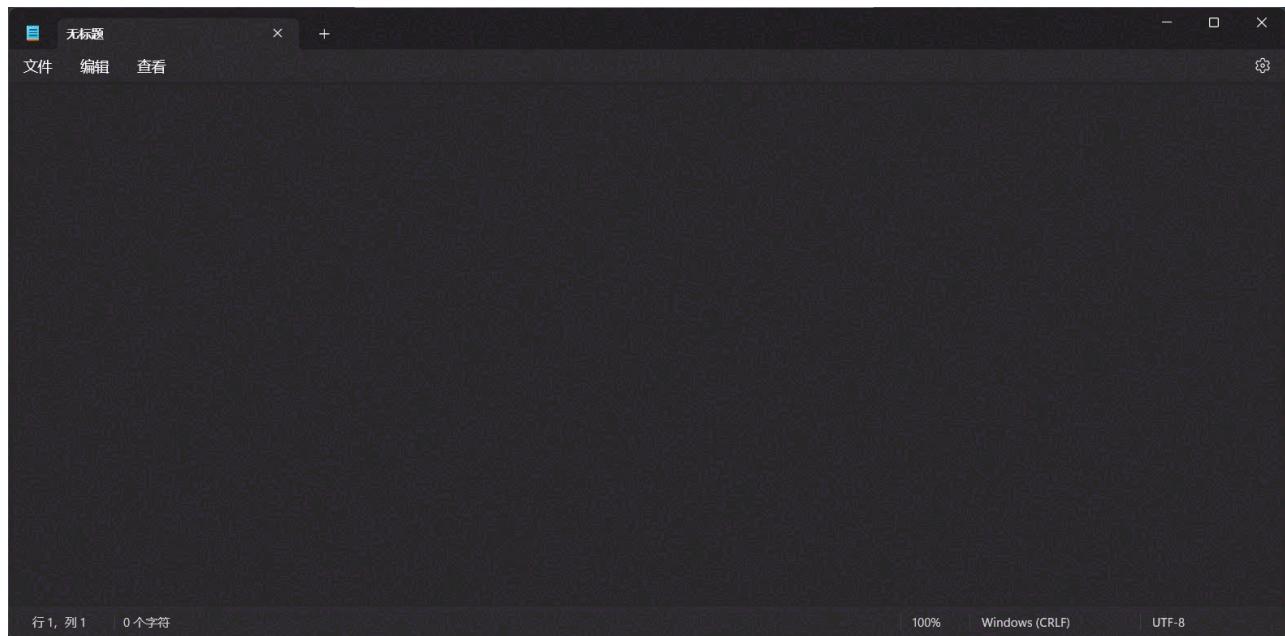
2.1 VSCode是什么的?

vscode简称code,他是一个ide(集成开发环境),但是说这个东西似乎不好理解,所以我们可以将它理解为一个**文本编辑器**.

你现在可以使用 `win+r` 快捷键打开一个叫做 `运行` 的窗口.随后,请在命令提示框中输入 `notepad`.如下图所示.



接下来按下 `Enter` 键,这里和你点击了 `确定` 键的效果是一样的.你会打开这样一个窗口.



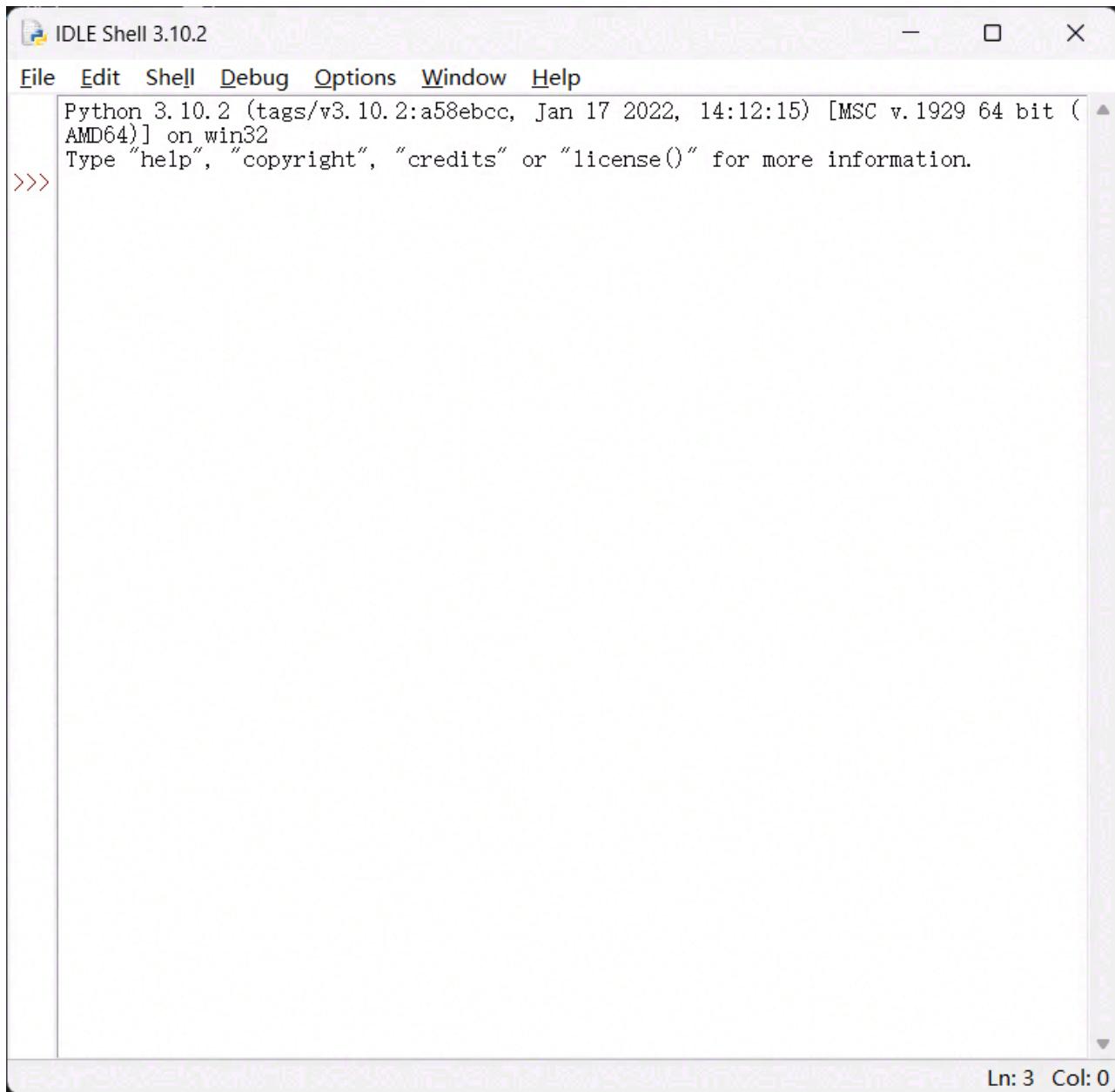
可能会有颜色不一样的情况,这是因为Windows的主题颜色不同,这并不影响使用.

这就是系统自带的记事本.它可以对文件进行 编辑 操作.

初步理解他和code就很是类似的.所以**VSCode可以用来编辑文件**.

2.2 为什么选择vscode?

既然code仅仅就是一个文本编辑器,那为什么我们要选择使用code,而不是notepad或者下载python时自带的ide呢?



如上图,这就是python自带的idle的shell窗口.(暂时不用纠结什么是idle,什么是shell,python部分我会告诉你他有什么用)

其实code的优点有很多,但是最主要的原因还是两个

1. 开源
2. 丰富的插件市场

开源的优点我暂且不讨论,随着你使用电脑的时间增长,你会越来越意识到开源的好处.

丰富的插件市场是一个很大的优点.

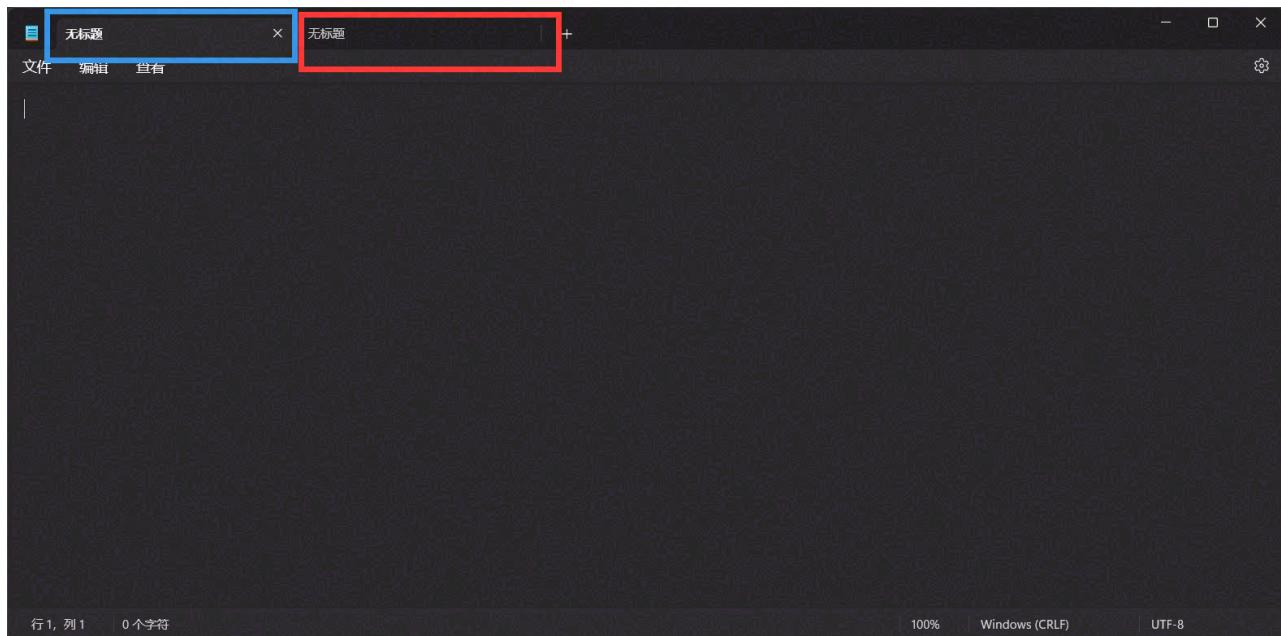
我们选择一个ide,是为了让自己写代码写的很舒服很轻松对不对?ide应该会减少我们的工作量,简化我们的操作对吗?所以使用code,我们可以下载简化我们需要做的很多事情,比如将终端集成到code中,代码补全,纠错,Debug等.这些概念我会在后面告诉你.

code相较于其他ide,他更轻量(体积更小),响应快,体验好.丰富的插件市场可以为你提供各种各样的稀奇古怪的功能,甚至如果你不满意,你还可以自己写一个插件来满足自己的需求,这毫无疑问是非常诱人的一件事情.

总之,使用code的理由就是它可以极大程度的减少我们的工作量,并优化我们的写代码体验.

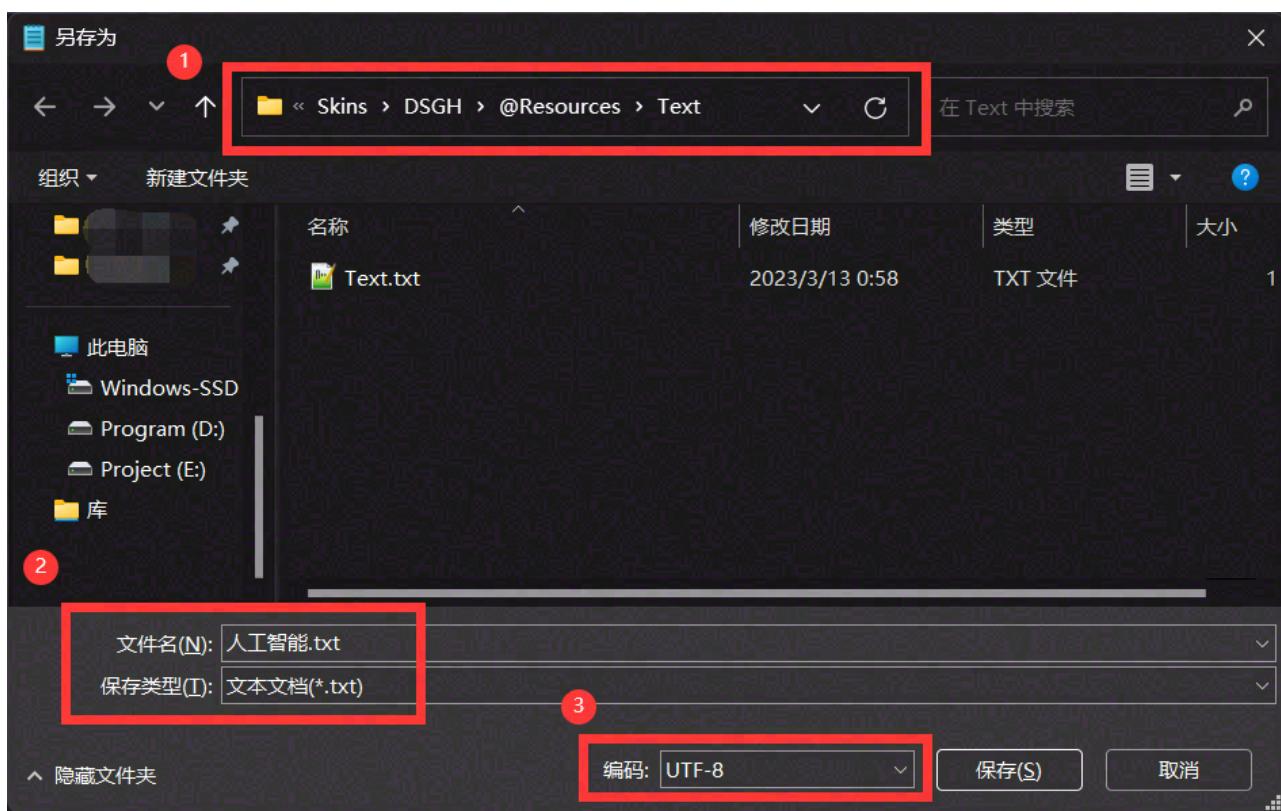
2.3 记事本的基本操作

和vscode类似,你可以使用 文件->新建标签页 或者使用 $\text{Ctrl} + \text{N}$ 的快捷键来新建一个标签页.



上图中的蓝色部分和红色部分就是两个不同的标签页.这里的标签页可以按照浏览器的不同标签页来理解.你可以分别在这两个标签页中书写内容,他们互相不会产生影响.

例如第一个标签页我书写了一个python,第二个标签页我书写了一个 人工智能 .我认为我现在已经编写完成了,所以我需要保存这两个文件.使用 文件->保存 或者 $\text{Ctrl} + \text{S}$ 功能来对文件进行保存.



这是当你按下 $\text{Ctrl} + \text{S}$ 后的界面,他会弹出一个窗口,这个窗口有以上三个部分需要注意.

1. 保存路径

这个部分叫做 地址栏 ,代表了资源管理器的地址.

这意味着你的文件保存在哪里,比如说你写好了作业,你是决定将他放在书桌的抽屉里还是再书桌上?正确的路径可以让你找到这个文件并再次进行处理.

2. 这里有两个需要注意的地方

1. 文件名

文件名会自动读取第一行的内容作为文件名,但是你同样可以修改他.值得注意的是文件名的格式 `文件名.后缀`.

`文件名` 代表了这个文件的名字,而 `后缀` 则代表了这个文件的类型.你可以修改后缀达到修改文件类型的目的.

2. 保存类型

这就是 `后缀`,初用电脑,我们可能不熟悉各个类型的简写是什么,如 `mp3` 是什么格式? `html` 是什么格式? `md` 是什么格式?当你无法再在上面的保存类型中的后缀中设置文件格式时你可以来到这个选项的下拉列表中选择文件类型.

记事本新建的文件默认是 `txt` 文件,也就是 `文本文件`.

3. 编码

这个东西很重要,但是现在你可能不会意识到,但是没关系,你现在只需要知道文件有编码类型这个东西就够了,同样的,可以在保存的时候进行修改.常见的有 `utf-8` 或者 `ANSI` 等.

截止到现在,你已经进行了 新建文件->保存文件 的操作了.接下来我们需要打开一个文件进行修改.

这里有两种方法.

1. 找到你需要打开的文件,随后选择打开的方式(软件).
2. 打开软件,然后在软件中打开你需要打开的文件.

首先是第一种.



找到了我们保存的文件,对着这个文件 右击 打开菜单.我们有两种方式打开.

第一种是直接点击第一个选项 打开.你同样可以按键盘上的`Enter`键来执行这个命令.

随后你会使用系统默认的软件来打开这种格式的文件,如果你的系统没有打开该文件的默认软件的话,他不会有这个选项.



此时你只能选择 打开方式 如下图,选择要给打开方式来打开了.



可以看到上图中我的打开方式有三种

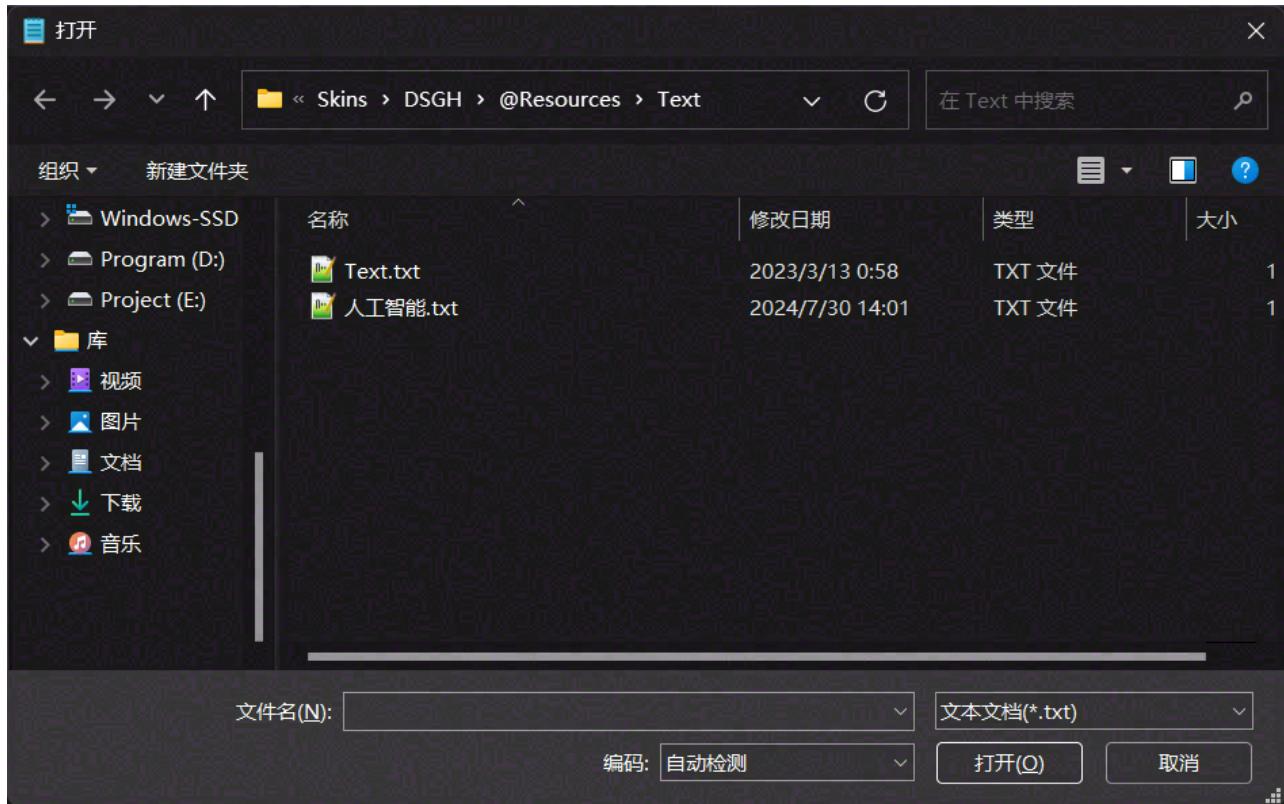
1. 一款叫做 NotePad++ 的软件
2. word
3. 记事本

在这里,你直接选择记事本就可以打开了.

选择其他应用的细节我就不细讲了.

值得注意的是,如果你仅仅是想打开一个文件,而不更改他的打开方式,你觉得默认的打开方式就很好,那么你可以直接双击文件本身,这和右键后的 打开 选项功能相同.

接下来介绍在软件中打开文件的方式,我们可以在 文件->打开 或者 $\text{Ctrl} + \text{O}$ 来打开文件.



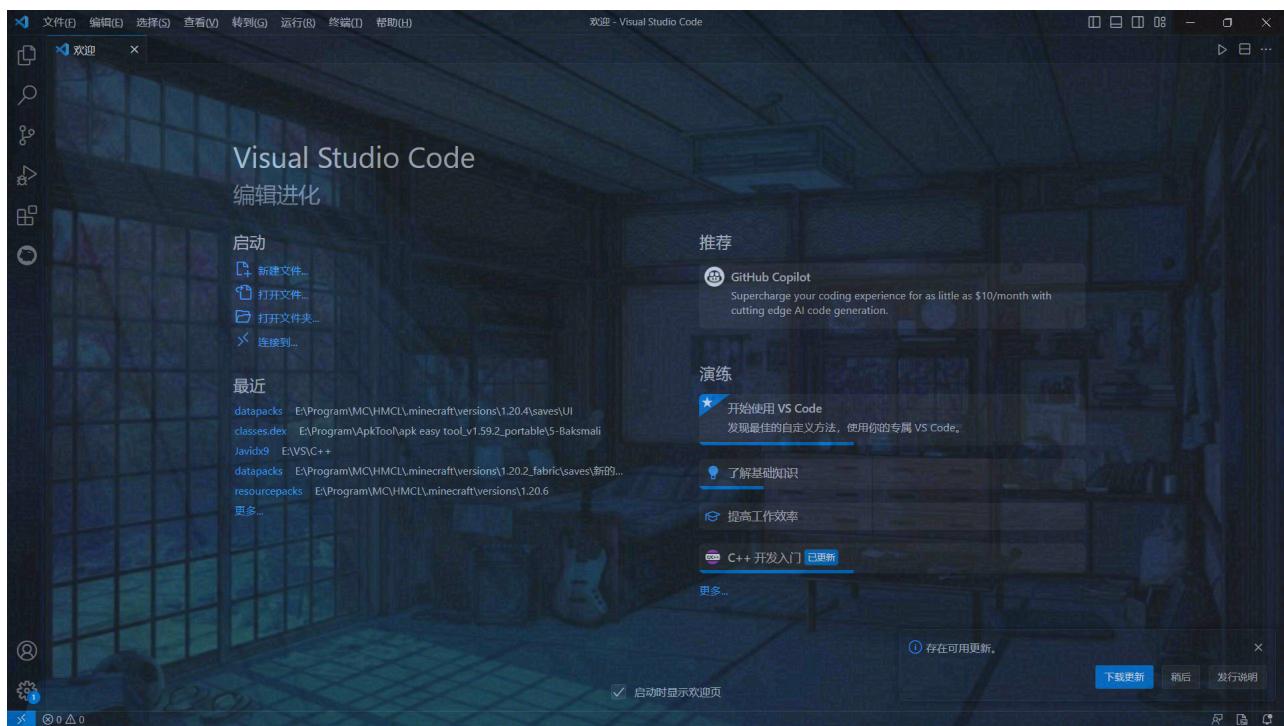
你会打开如上的一个窗口,和保存文件的时候很像,不同的是下方的选项区,接下来仅仅只需要选中你需要打开的文件就好了,随后点击 **打开** 即可.

2.4 VSCode的基本操作

接下来我们来看VSCode.

我的code的版本是 1.81.1.这并不是最新版本,但是整体操作大差不差.

如果你是第一次使用这个软件,或者说你之前没有打开过文件,那么界面应该如下所示.

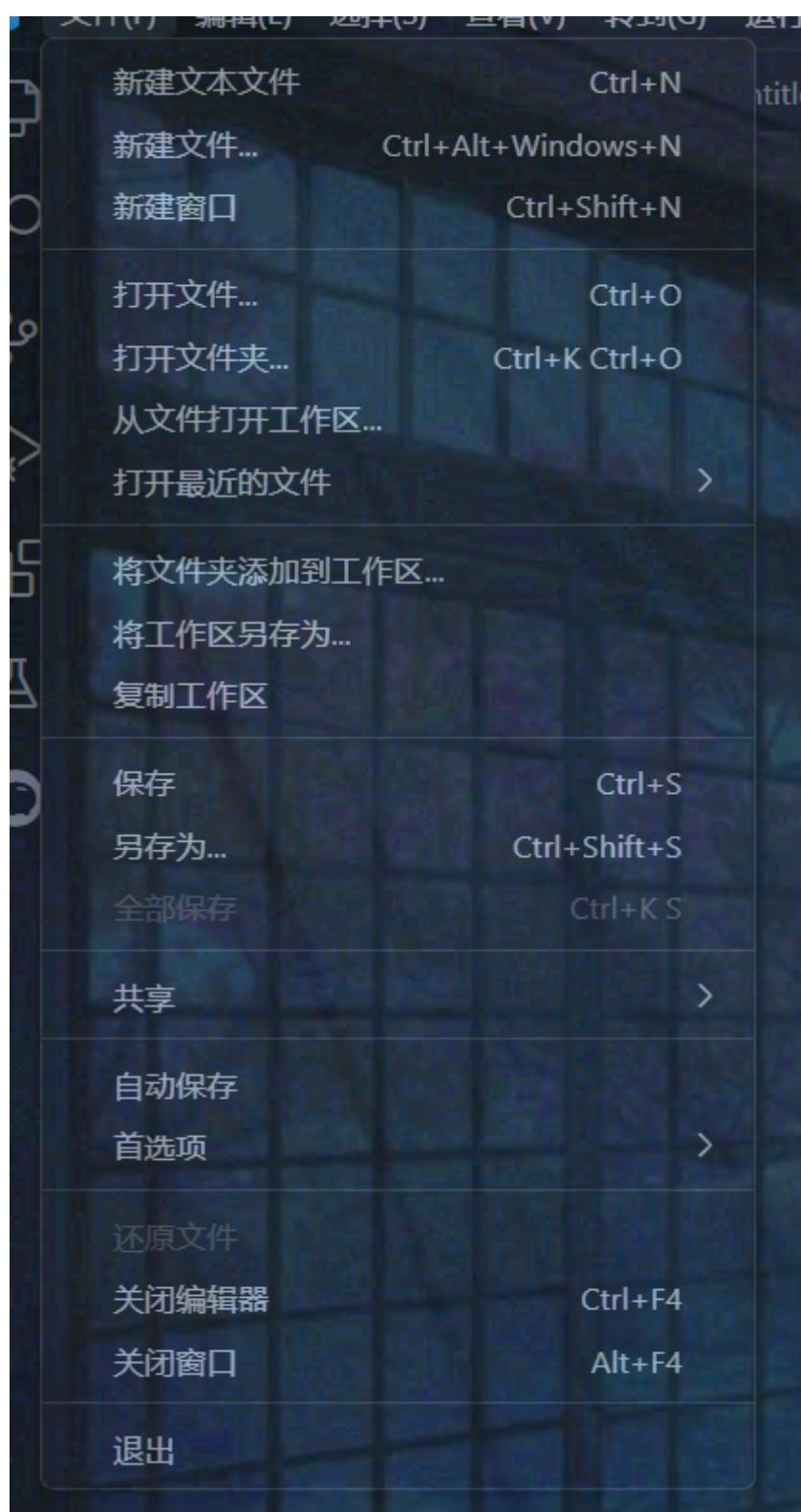


我们的差别可能在于我有一个图片当作背景,以及我们的欢迎这个标签页内的东西不同.

昨天晚上你已经安装了两个插件 汉化插件 和python插件,现在code就可以写python了.

2.4.1 新建文件

和记事本的流程一样, `文件->新建文件` 这里我们发现他有两个选项.



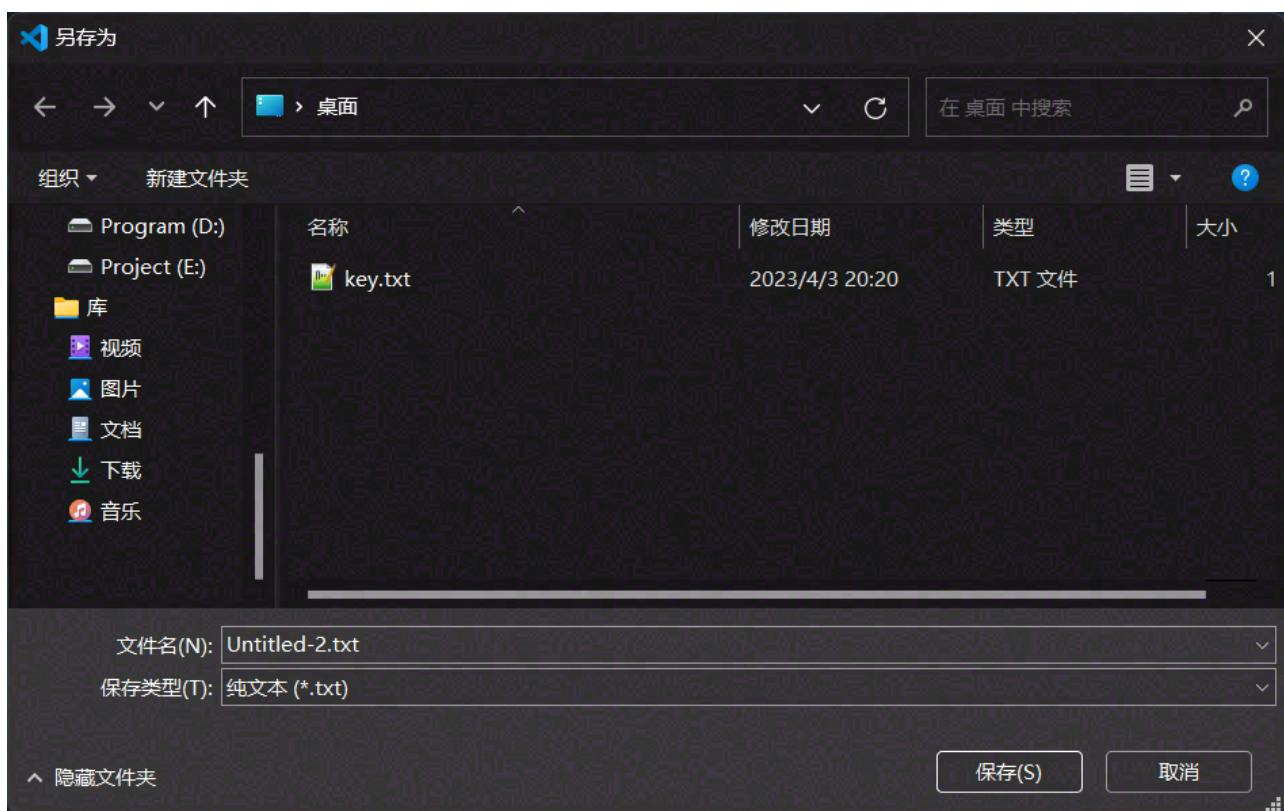
新建文本文件 和 新建文件 .

对于第一个很好理解, 他就是新建了一个文本文件, 也就是 `.txt` 文件, 他和在记事本中新建的文件没有区别.



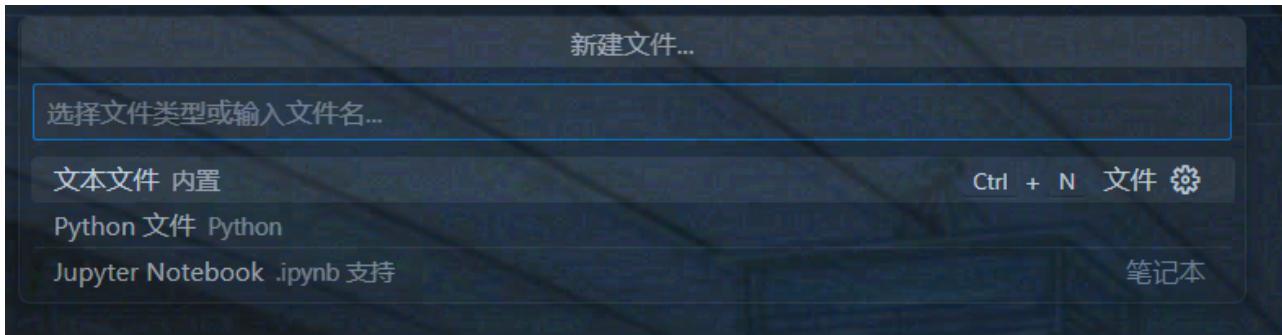
提示:上方的蓝色字体都是可以点击的.

我们在下方发现它显示了如下几个信息 行列数 空格数 编码格式 行尾序列 文件类型,我们观察到他的文件类型是纯文本 txt 编码格式是 UTF-8 .很熟悉是不是?

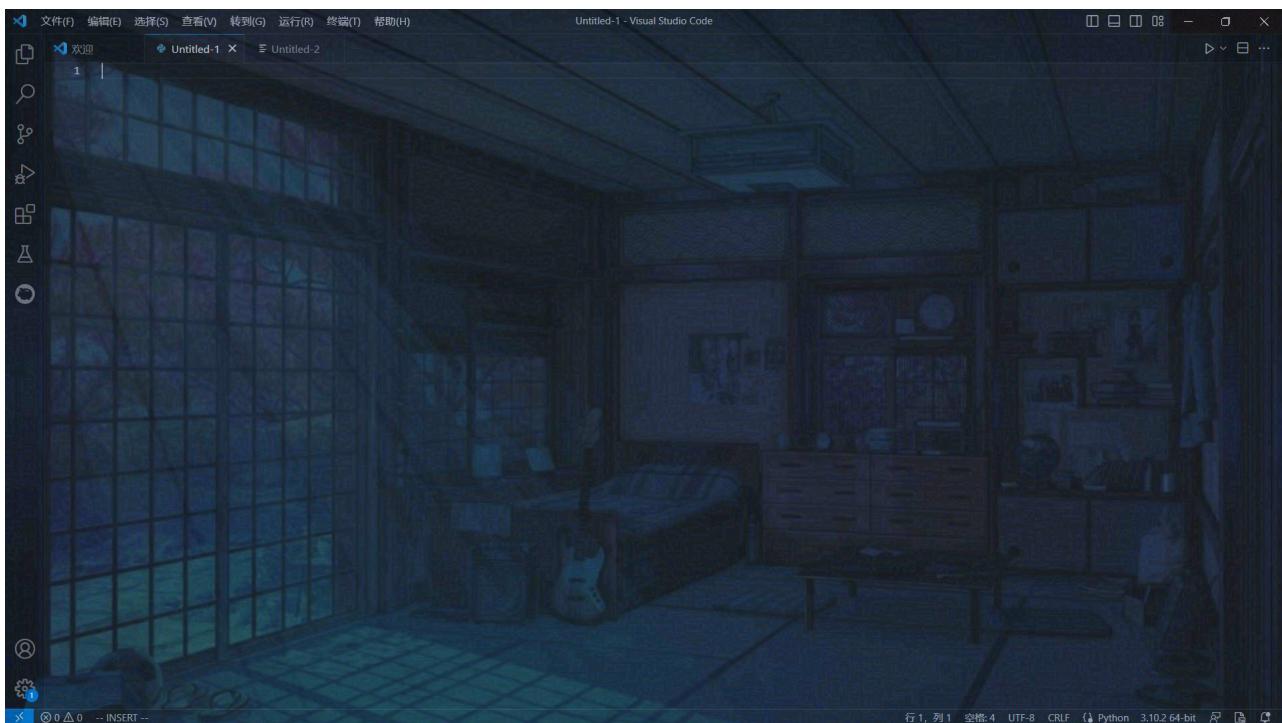


不过这样一来,他的保存界面就没有让你修改编码的地方了.保存的操作我已经说过了,如果不记得,请回到 2.2 记事本的基本操作 去看.

对于新建文件,你会出现如下弹窗,可以选择你需要新建的文件格式.



如上,我们选择第二个 Python 文件 即可.



观察下方的信息提示区,我们发现他的文件类型是python,而后面所跟的一串数字则代表了解释器版本,也就是你安装的python版本.在后面的终端操作部分我会告诉你如何判断我们的python版本.

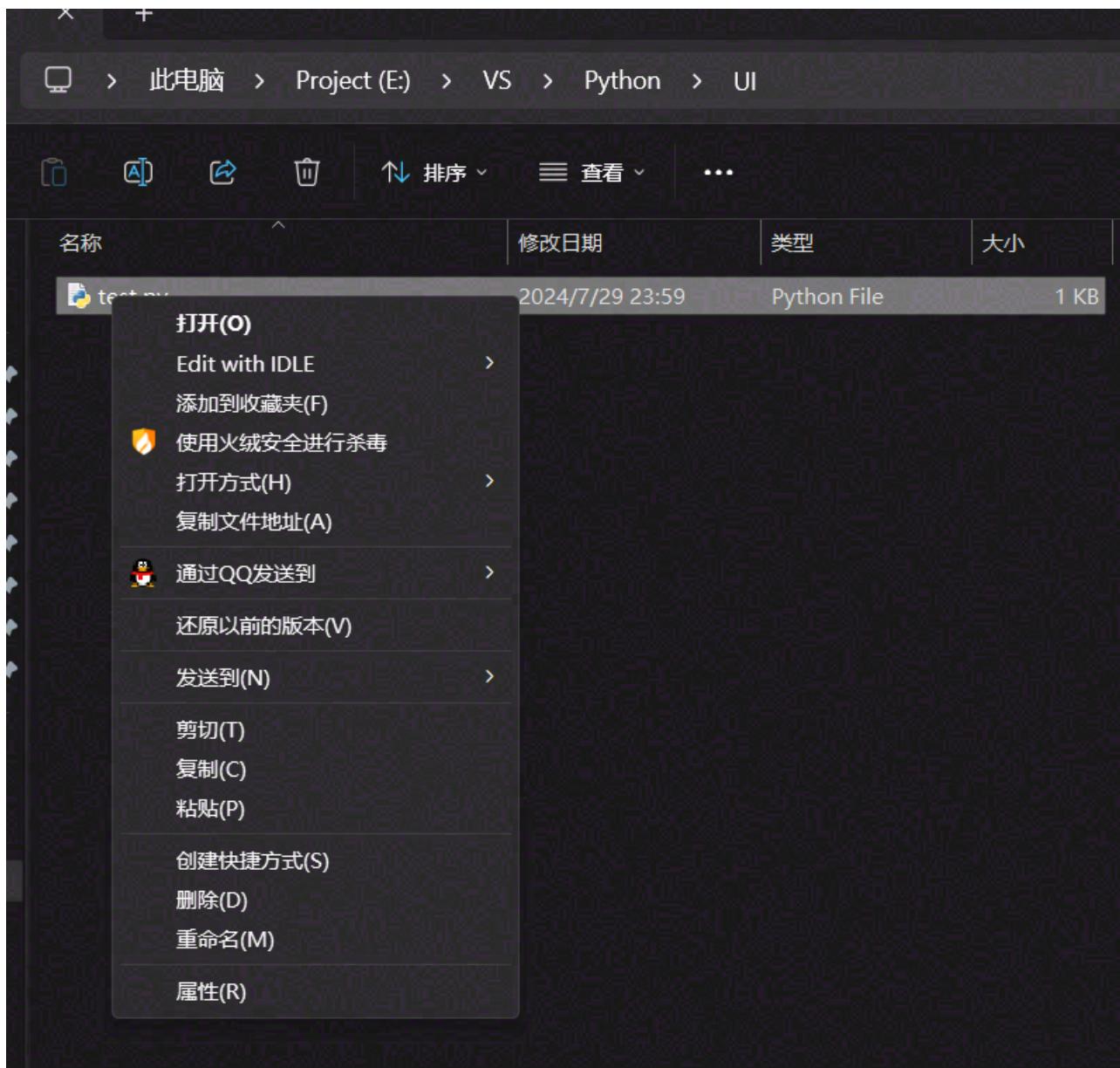
随后你就可以在这个文件中进行编辑了.我们以之前我发给你的代码为例.将其粘贴在文件中,并保存在桌面上,命名为 `HelloWorld.py`.

经过这样一个过程,是否发现和记事本的操作如出一辙?

2.4.2 打开文件

同样的方式我们也可以打开文件,在code中打开一个python文件很简单,我认为我不需要细讲了,如果有不会的,再问我吧?

重点讲一下如何在资源管理器中使用code打开一个文件.



首先,找到正确的路径,也就是你的文件保存在了哪里.随后对文件进行右键,这里有三种方式打开这个文件.

1. 直接 打开 也就是和双击一样的效果

他会使用默认的打开方式来打开对应的文件,所以这一步类似于执行了这个程序文件,这里有一个关于python的小细节,我会在后文中讲到.

2. Edit with IDLE 选项

这个选项会使用下载python时候自带的ide来打开文件



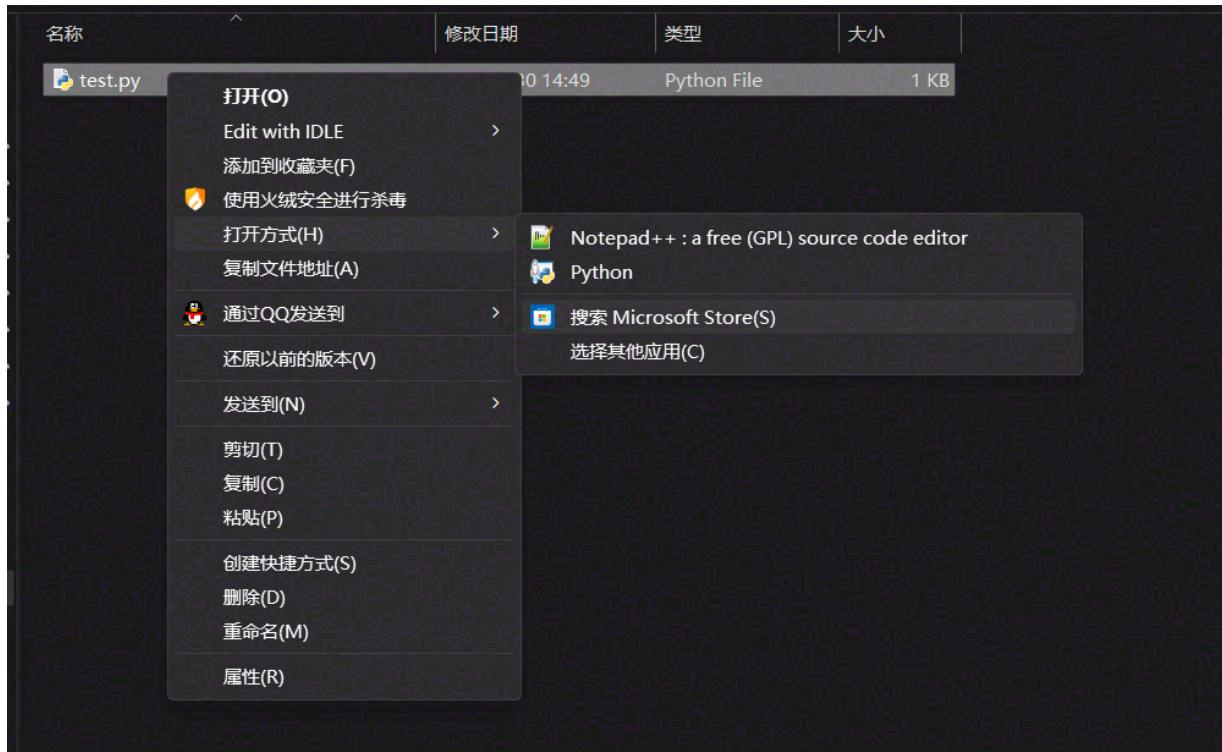
也就是旁边的这个.

A screenshot of a Windows-style application window titled "test.py - E:\VS\Python\UI\test.py (3.10.2)". The window contains a menu bar with File, Edit, Format, Run, Options, Window, and Help. The main area displays Python code for a Tkinter application. The code imports tkinter, creates a window titled 'DSGH', sets its geometry to 300x300, defines a callback function to change a label's text to "Hello World!", creates a yellow background label with black text, and adds a button labeled "执行" (Execute) that calls the callback. The status bar at the bottom shows "Ln: 1 Col: 0".

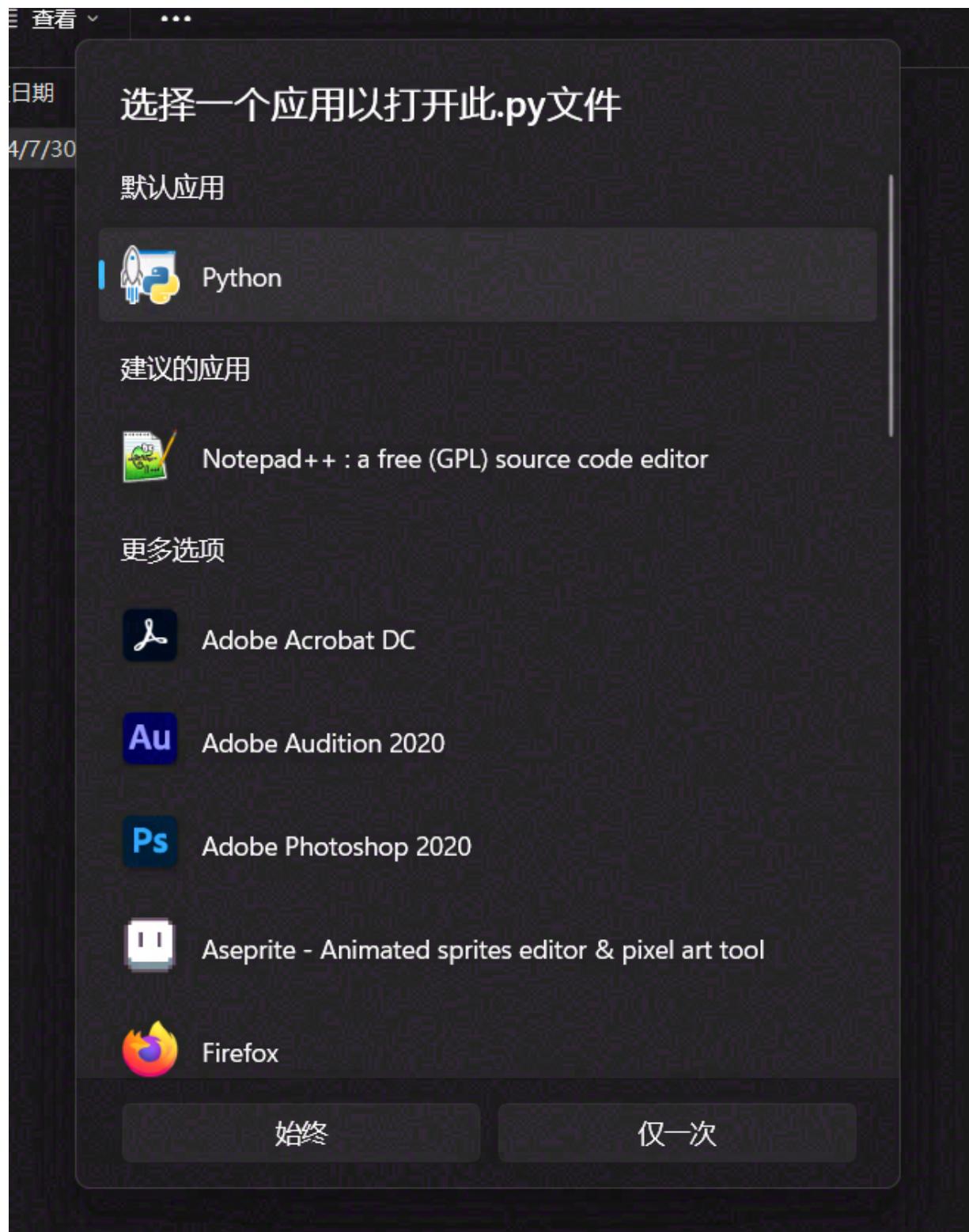
```
import tkinter as tk
window = tk.Tk()
window.title('DSGH')
window.geometry('300x300')
def callback():
    text.configure(text="Hello World!")
text=tk.Label(window, text="Hello", background="yellow", fg="black", font=('Times', 1))
text.pack()
button = tk.Button(window, text="执行", command=callback)
button.pack()
window.mainloop()
```

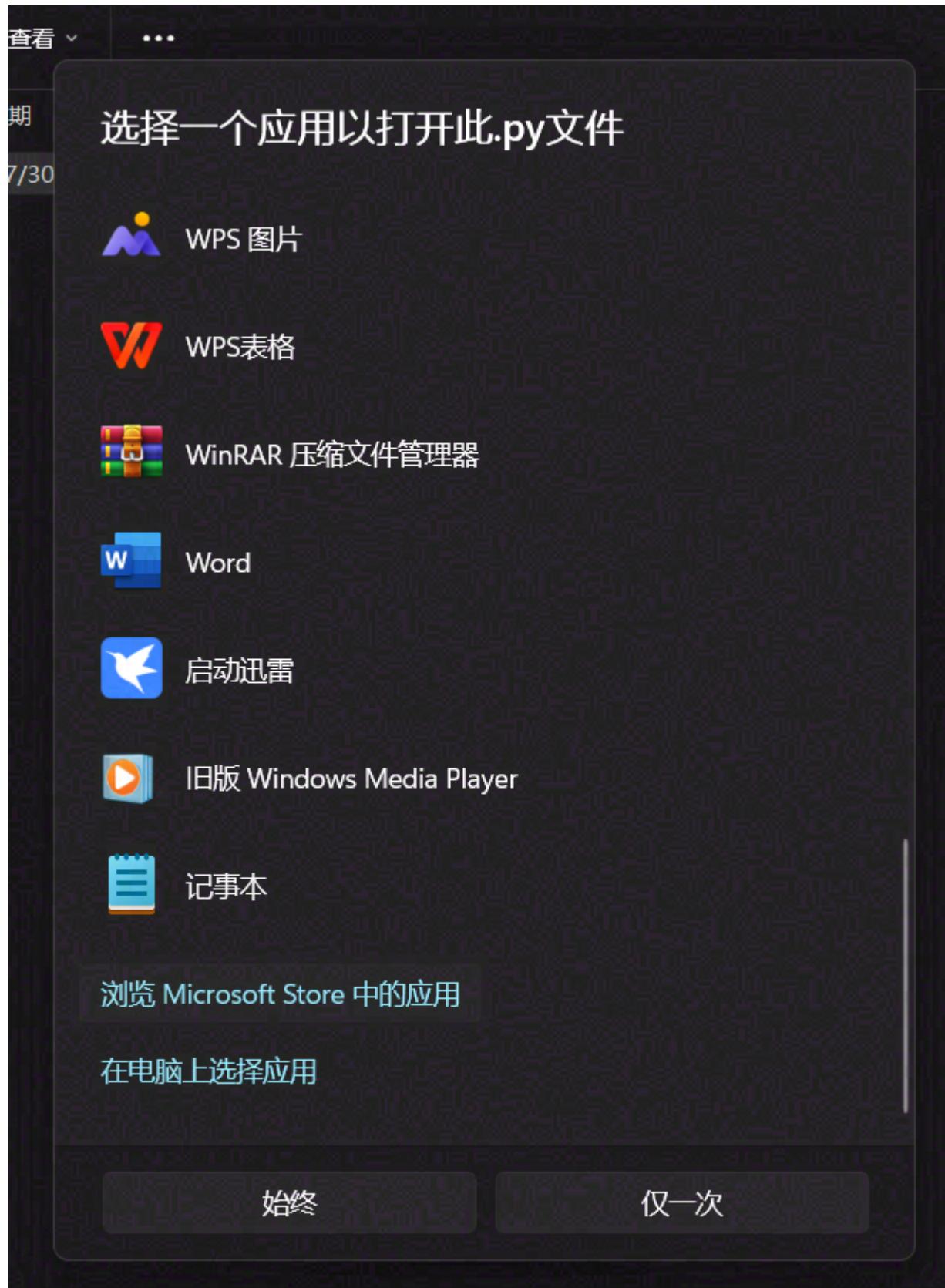
这是打开后的窗口,这个窗口实际上也可以代替vscode,它也可以正常的编写代码,但是他的体验并没有code那么好.

3. 打开方式

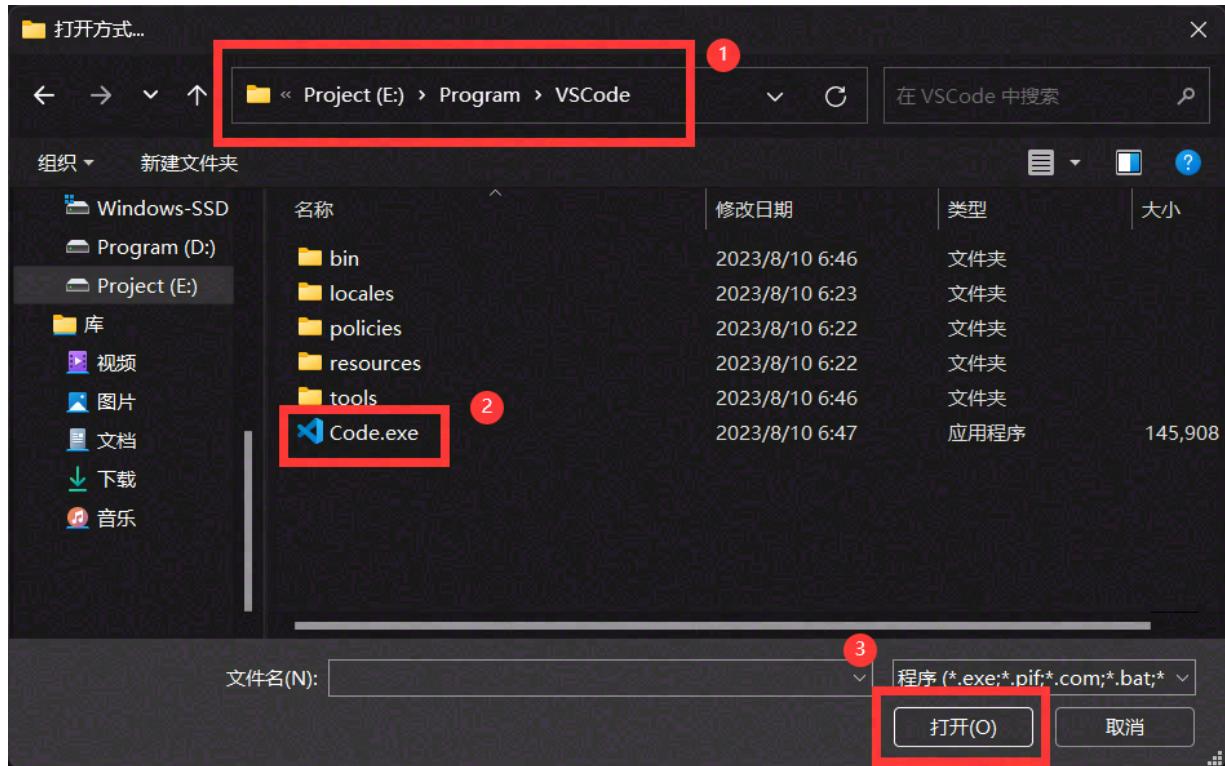


右键并选择了 打开方式 后,我们发现并没有code出现,所以我们需要点击 二级菜单 中的 选择其他应用 这个选项.



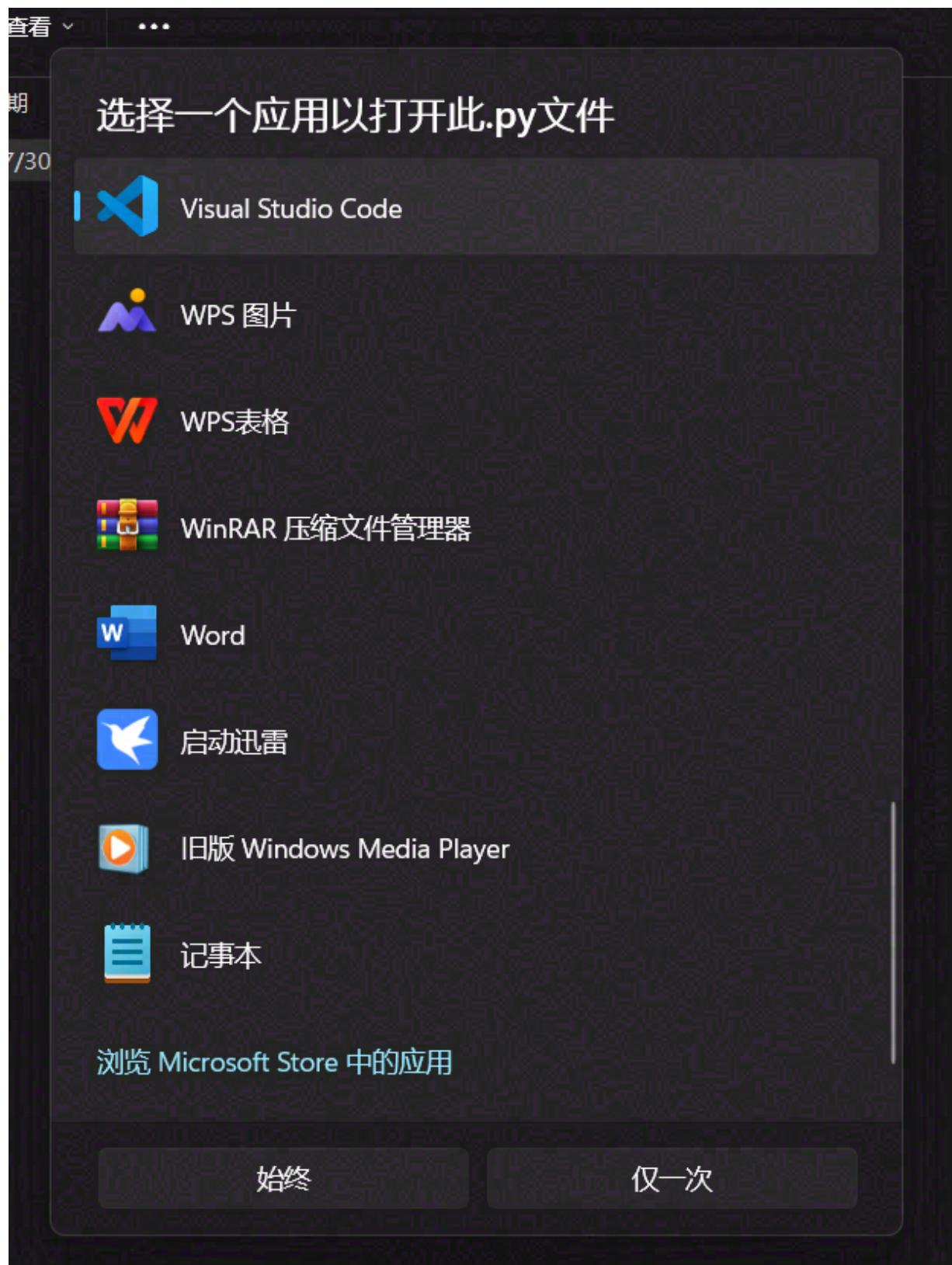


你可以看到上面这样一个菜单,我们菜单中的内容可能并不一样,但是这并不重要,你可以在这个菜单中寻找vscode这个软件,如果没有找到,你应该点击[在电脑上选择应用](#).

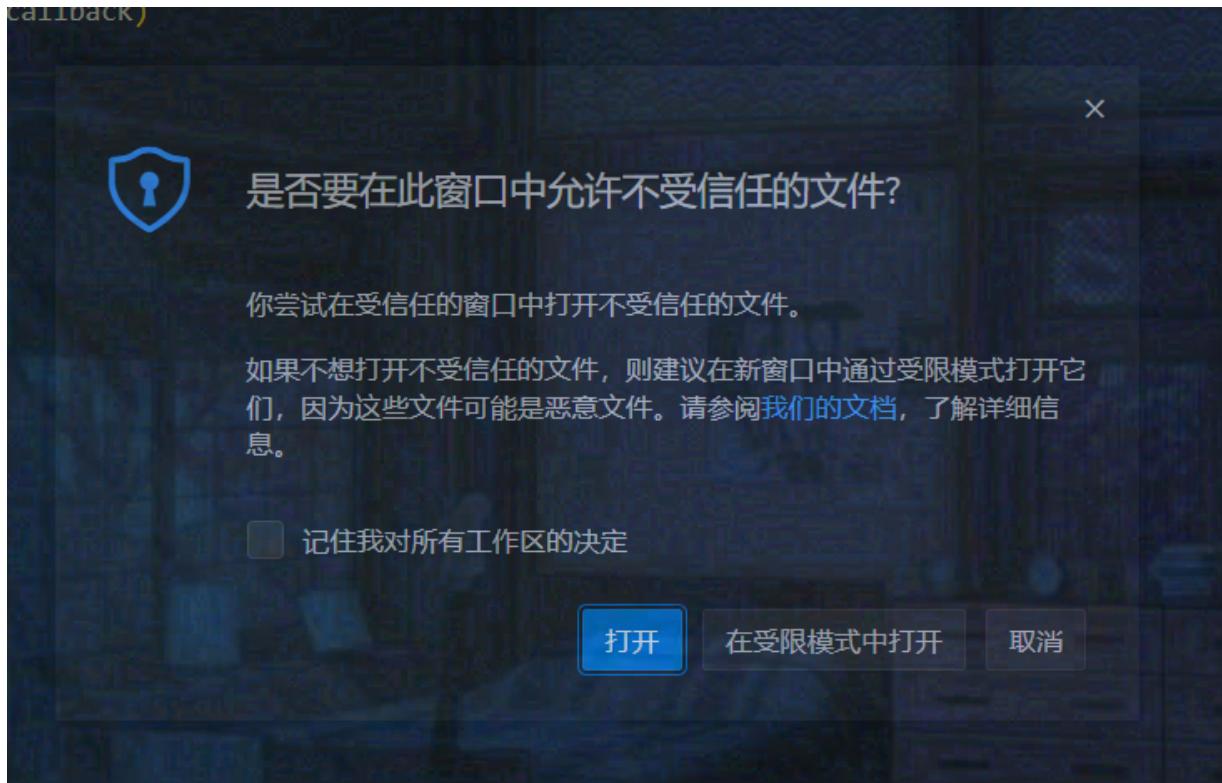


接下来,这个窗口你已经很熟悉了,找到正确的地址,也就是 vscode 的安装路径.

选中 code.exe 并打开.



接下来,code会出现在列表中,选中并点击 仅一次 即可使用code打开这个文件了.

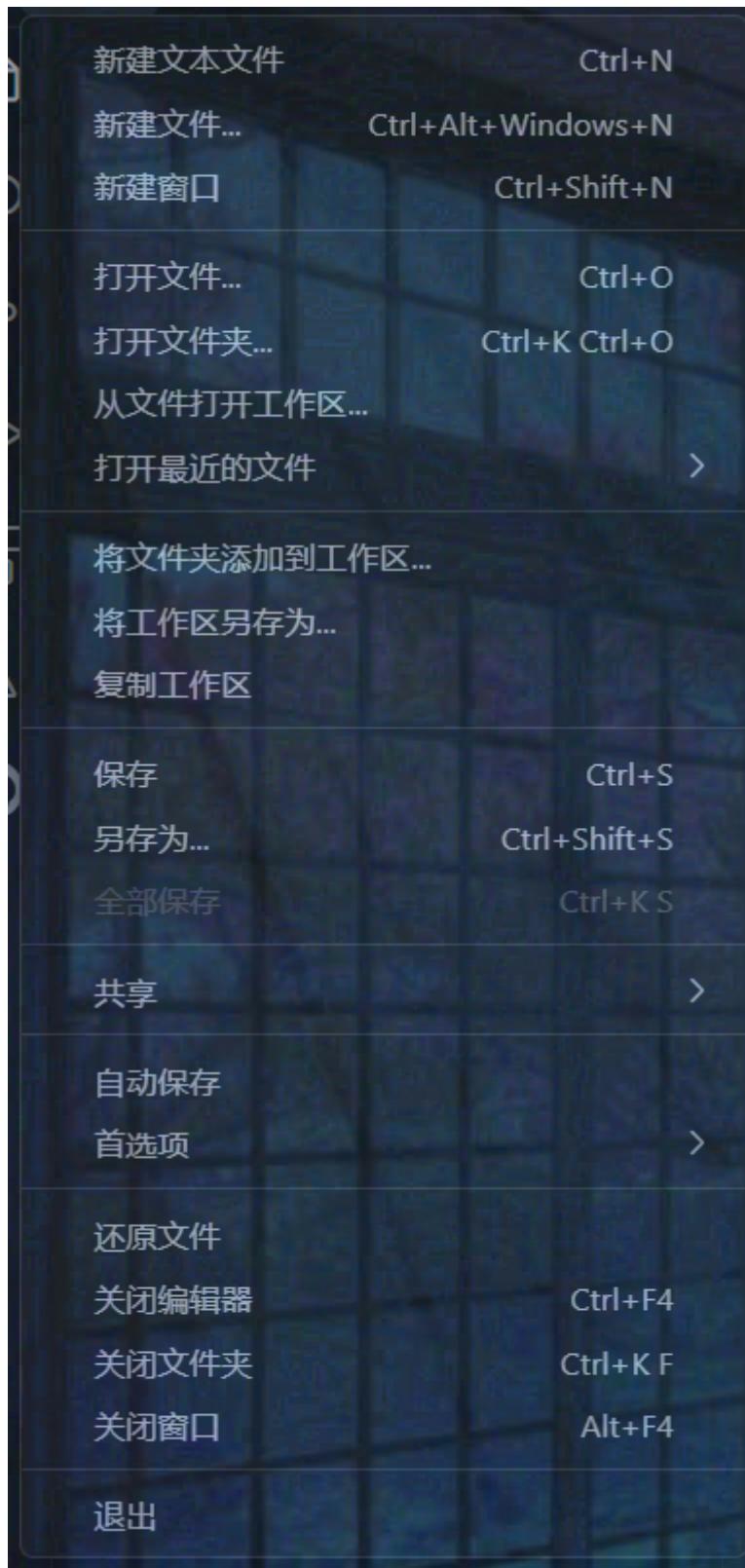


打开文件code会提醒你这个文件并不受信任,由于这个文件是我们自己创建并编写的,所以理所当然的我们很清楚他是很安全的,所以我们选择打开即可.

这样你就可以看到文件被vscode打开了.

2.4.3 打开文件夹

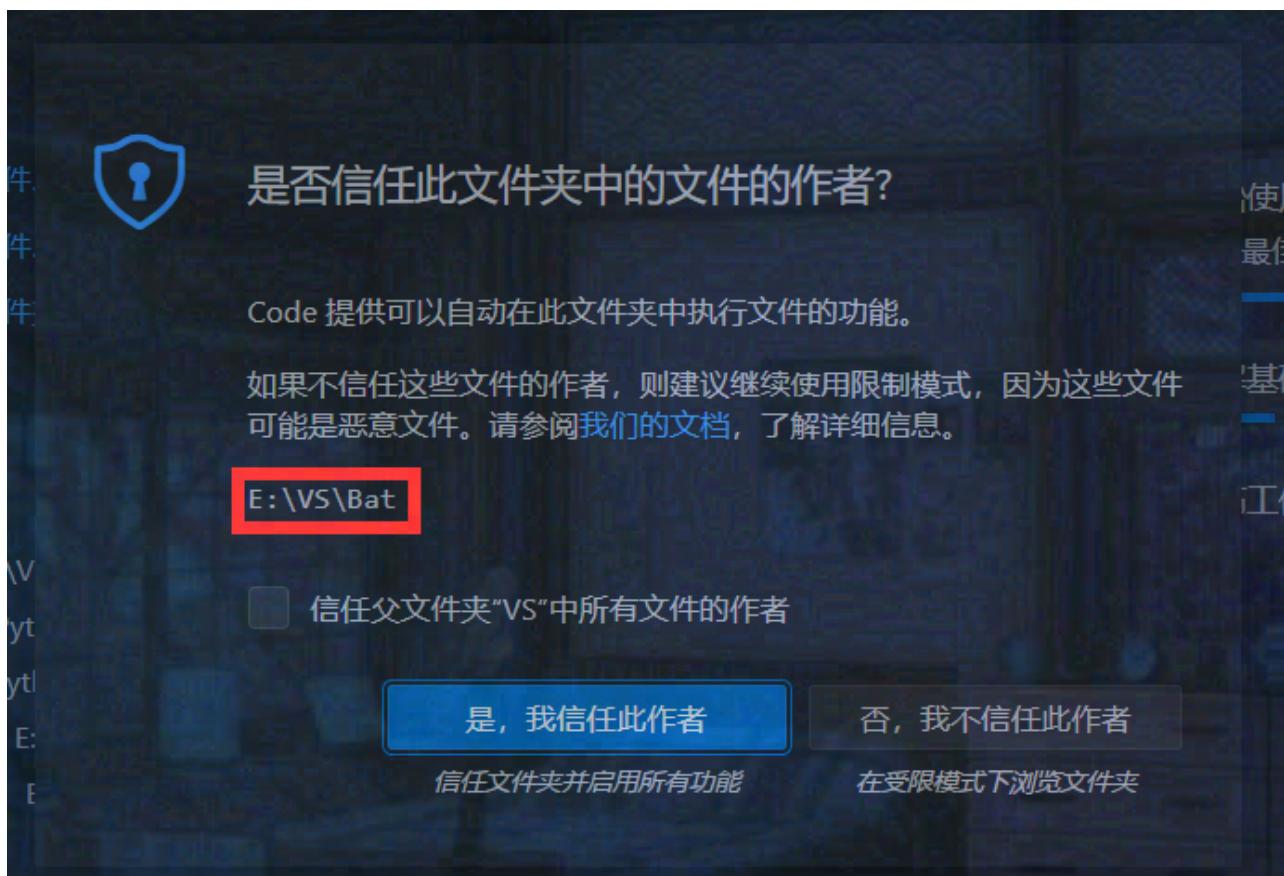
在实际的开发过程中,我们很容易理解,一个程序并不仅仅由一个文件构成的,他理所当然的可能会由无数个文件夹套文件夹组成,所以为了方便管理,code提供了打开文件夹的功能.



在文件选项中,点击 打开文件夹 选项



这个窗口和打开文件的窗口十分像,你只需要在这里打开你需要打开的文件夹,并选择文件夹即可.



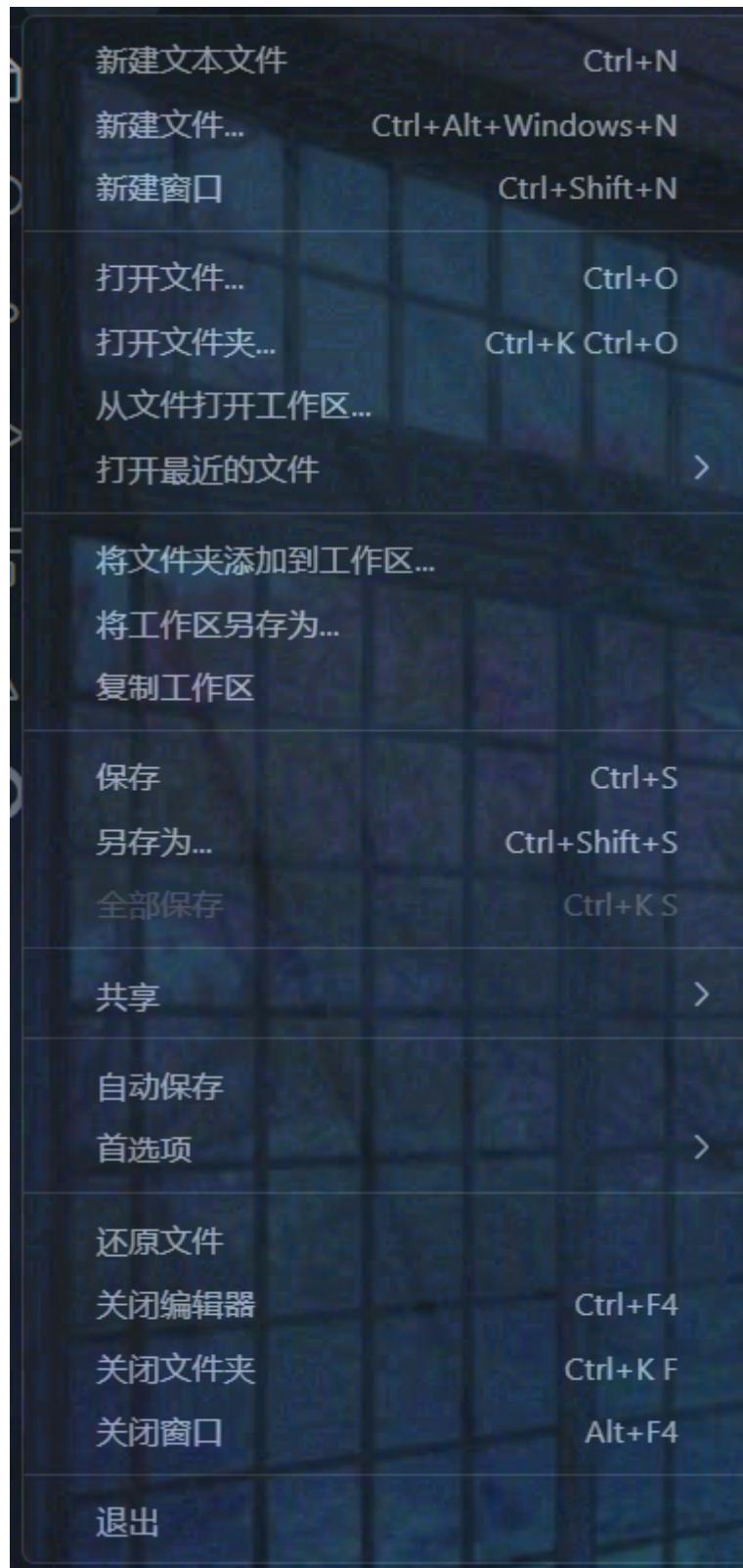
第一次打开文件夹,code会提示你是否相信这个文件夹,由于我的python已经不是第一次打开,所以我是用别的文件夹来当作例子.

上图中画红框的地方就是这个文件夹的路径,而这个文件是我们创建的,所以哦我们选择是,值得注意的是,我一般也会勾选 信任父文件夹... 这个选项.因为我的文件分类后,我很清楚这个文件夹没有问题.

2.4.4 快捷键

在code中,大部分操作都有快捷键,我也非常建议你经常使用快捷键,比如打开文件你可以直接按 `Ctrl+N` 而不是用鼠标去点击.

大部分功能的快捷键会显示在选项后面,如下图所示.



不过想必你也一定注意到了有两个快捷键与众不同,正常的快捷键都很好理解,如打开文件 `Ctrl+N`,毫无疑问这意味着按下 `Ctrl` 同时并按下 `N`,

那么打开文件夹 `Ctrl+K Ctrl+O` 是什么意思呢?它意味着你需要依次按下 `Ctrl + K` 和 `Ctrl + O` 两个快捷键.

这里我们就可以理解 + 连接的两个按键意味着需要同时按下,而使用空格分隔的两个按键意味着需要依次按下.

那么我们可以再看一看关闭文件夹这个快捷键 `Ctrl+K F`,很好理解,这意味着这个按键分两次按下,第一次是按下 `Ctrl` + `K`,第二次仅仅只需要按下 `F` 按键.

tips:

1. 如同 打开文件夹 一样的快捷键,它是由两个 `Ctrl` 前置按键组成的,再分开按下 `Ctrl` + `K`、
和 `Ctrl` + `O` 这两个按键的时候,我们可以不松开 `Ctrl`,也就是说 `Ctrl+K Ctrl+O` 等价于 `Ctrl+(K O)`,
也就是按住 `Ctrl` 不松手,依次按下 `K` 和 `O` 按键.
2. 尽管在快捷键提示中 `Ctrl` + `A` 中的 `A` 是大写,这并不意味着我们需要按下 `Shift` + `a` 来将 `a` 转换成为 `A`.
可以参考 新建窗口 这个快捷键 `Ctrl+Shift+N`,他就使用了两个前置按键 `Ctrl` 和 `Shift`.
3. 关于前置按键,一般在Windows中指的是 `Ctrl`, `Shift`, `Windows`, `Alt` 这四个按键,但是实际上所有的按键都可以作为前置按键,不过Windows自带的大部分快捷键都是使用这四个按键作为前置按键的.
比如 `win` + `I` 打开设置.

2.4.5 ToDoing

实际上code还有很多东西我想要和你说,但是考虑到你的电脑基础知识并不足,哪怕我和你说了或许并不会对你产生帮助,所以这部分是 todoing...

等到你有了一定的电脑操作之后,如果你还需要我帮你的話,可以来找我,我会把这部分完善.

2.5 授人与鱼不如授人与渔

Code的功能强大,内容复杂繁多,如果我一个一个功能的说,肯定是不行的,更何况我也不敢说我会code的所有功能,那么我们究竟该如何理解code这个软件呢?实际上,这里使用28定律.我们80%的时间都在使用一个软件的20%的功能.

实际情况也是如此对于code,在使用过程中,我们应该抱有常用常新的态度,经常使用,便会经常发现新的东西,而这些新的东西并不是来源于某些教程,而是来源于你自己的使用体验.因为每个人的使用情景不用,使用要求不同,以我为例,实际上我是用code写datapack会很多,但是如果写cpp或者c#,我会使用vs而不是code.vs你可以理解为功能更加齐全的code,但是相应的,他也会上手难度更高,体量更大等.甚至写python我很多情况下都是使用终端去写的,所以每个人的使用场景并不相同,基于你的使用场景你会学到独属于你的使用技巧和方法,随后你也可以随着时间的流逝总结出自己的方法论.

那么为什么会有那剩下的80%的功能会被写入软件呢?

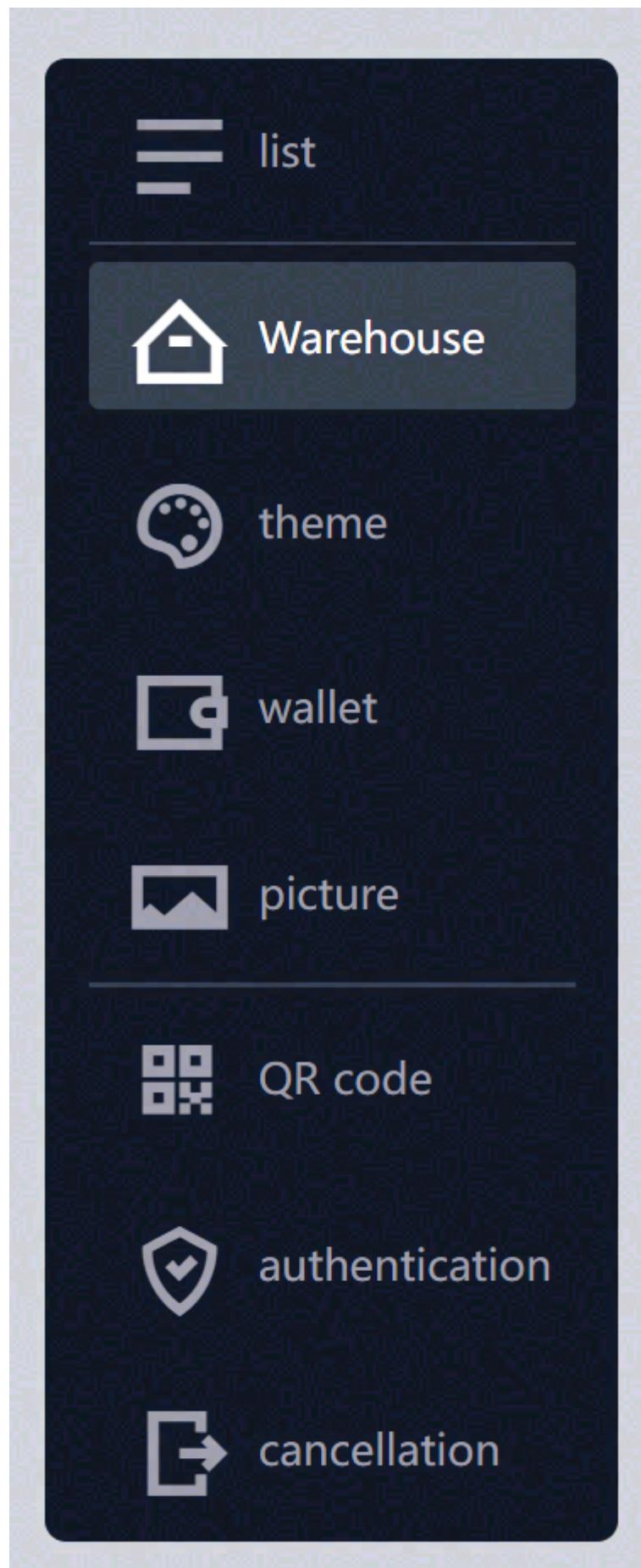
原因很简单,总有人需要用到.

以多行编辑为例,多行编辑指的是在code中可以同时对多行进行编辑,比如你在第一行中输入了一个"A",那么在你选中的这些行中,他都会输入一个A.如果表现在编辑界面就是这些行上都有光标在闪烁.

如果使用code的人是一个写前端的人,也就是html的人,他经常需要用到同时修改很多行的内容的时候,比如如下的 html 代码就可以看出来.

```
1 <i class="iconfont icon-liebiao"></i><span>list</span>
2 <i class="iconfont icon-cangku"></i><span> warehouse</span>
3 <i class="iconfont icon-zhuti_tiaosepan"></i><span>theme</span>
4 <i class="iconfont icon-qianbao"></i><span>wallet</span>
5 <i class="iconfont icon-tupian"></i><span>picture</span>
6 <i class="iconfont icon-erweima"></i><span>QR code</span>
7 <i class="iconfont icon-dunpaibaoxianrenzheng"></i><span>authentication</span>
8 <i class="iconfont icon-dengchu"></i><span>cancellation</span>
```

以上这段代码是用于写一个 icon 的列表.



这是它的实现效果.我们观察到代码部分 `<i class="iconfont icon-` 是完全一样的,而后面确有部分不一样.但是都可以简化为如下结构.

```
1 | <i class="iconfont icon-不同处1"></i><span>不同处2</span>
```

所以我们可以以以下的方式输入这8行.

那就是在第1行输入 `<i class="iconfont icon- "></i>`. 并使用code的自动复制功能复制到2到7行去.

当然你可能会疑问为什么不把以上的内容复制8份呢? 原因其实很简单, 在于修改方便.

```
1 | <i class="iconfont icon-tiebiao"></i><span>list</span>
2 | <i class="iconfont icon-cangku"></i><span> warehouse</span>
3 | <i class="iconfont icon-zhuti_tiaosepan"></i><span>theme</span>
4 | <i class="iconfont icon-qianbao"></i><span>wallet</span>
5 | <i class="iconfont icon-tupian"></i><span>picture</span>
6 | <i class="iconfont icon-erweima"></i><span>QR code</span>
7 | <i class="iconfont icon-dunpaibaoxianrenzheng"></i><span>authentication</span>
8 | <i class="iconfont icon-dengchu"></i><span>cancellation</span>
```

我们想要把所有的 `icon-` 更改为 `ico.`, 首先我们肯定不可能使用全局替换功能, 因为面对浩如烟海的代码, 我们不知道哪里还有 `icon-` 随意的替换可能会是网页出现问题. 那么局部替换呢? 我们可以点击8次, 来实现替换, 当然可以, 不过使用同时编辑8行的功能就会化简很多.

此时我们只需要使用鼠标将这八行一拖动, 随后将第一行的 `icon-` 修改为 `ico.` 即可.

但是如果是pyth呢? 实际上, 在软件中, 我们基本上不会有这种多行相同的语句出现, 原因很简单. 程序中存在一个东西叫做循环, 所以我们不需要使用穷举法穷举所有的 `icon`. 以及当我们需要大范围的实现某个相同的东西时, 我们可能会使用结构体, 并将使用循环的方式将实体化等.

这里出现了很多概念**结构体, 穷举, 实体化**等, 这些概念你会在以后学到, 这些东西也并不复杂, 如果你想学, 可以搜一下, 就很简单的可以学会了.

在使用code中, 我们只需要保持遇到问题, 解决问题的这样一个习惯, 就会逐渐掌握code的常用功能, 而剩下的那些生僻的功能, 也许确实可以极大程度的优化你的操作流程, 那你可以在长辈们的电脑上学习到, 一般来说使用软件很久的人, 都会有自己的经验总结, 我们可以通过阅读他们的经验总结, 来获取已经成熟的操作理论.

综上所述, 我们需要做到的是**常用常新理论, 遇到问题解决问题思维, 以及充分的交流**. 我们就会逐渐学会并掌握这个软件.

3 Python的基本知识

好的,现在我们来学习Python. Python我是当作一个玩具去学习的,所以我不太可能说给你讲的非常好,但是我会和你分享关于编程的基础常识和一些简单的语法.最后我在会分享一些我自己的学习经验和感悟,因为是我个人的看法,所以请你辩证的看待,也许在你系统性的学习之后,**会发现我说的是错误的.**

你会这里学到什么?

1. Python最基础的语法
2. Python在windows终端中的使用
3. 编程的基本思路

请注意:

3.1部分的代码,如果没有特殊说明,均为cpp,不要试图在python中输入它!

自3.2部分后,代码均为python.

3.1 关于编程的基本知识

在这里我不想分享所谓的术语,如果你想要知道这些东西,那你完全可以上百科去搜,你可以看到官方明确的解释.

3.1.1 什么是程序?

在我看来,程序是为了**满足人的需求**而产生的事物.

我们可以简单的将程序理解为软件,就好像是你桌面上安装的各种软件一样,qq,网易云音乐等等,他们都是软件,实际上,我们使用的Windows系统本身,他也是一个程序.不过是一个复杂得多的程序.

3.1.2 程序是干什么的?

截止到目前我接触的所有程序,都是对输入进行操作,并产生特定的输出的.



以上例子中,处理部分就是程序的任务,也是我们编写的代码需要负责的部分.

以做菜为例.

1. 输入

我们需要做什么菜?我们有什么食材?我们可以使用什么厨具?

这分别代表了**目标,资源,工具**.

2. 处理

我们使用厨具对食材进行加工,比如使用刀把蔬菜切开,削皮等等.随后使用锅对蔬菜进行烹饪.这个过程就是处理.

3. 输出

输出当然就是一盘做好的菜了.

上面的例子相比可以很好的立即程序需要做什么,在做菜这件事情中,厨师就好像是程序员一样.

接下来举两个程序的例子.

3.1.2.1 输出从0到n

要求是我们输入一个数字n,然后程序会自动输出从0到n之间的所有数字,包括n.

我们来分析上面这个例子.

1. 输入是什么?

输入是数字n.

2. 处理呢?

我们要想办法让程序从零开始输出到n.

3. 输出呢?

一个从零开始到n结束的序列.

好的,那么我们就可以得到下面的代码.现在我们不关心代码怎么得到的,后面我会以他为例进行讲解.

以下代码为python版本.

```
1 # 处理部分
2 def printNum(n):
3     for i in range(n + 1):
4         print(i)
5 # 输入部分
6 n = int(input("请输入一个数字 n:"))
7 # 输出部分
8 printNum(n)
```

这个代码实际上还是有问题的,后面我会改进他.

3.1.2.2 记录点击次数的按钮

我们想要个窗口,用户点击按钮,窗口上会显示用户点击按钮的次数.

使用同样的方式分析这个程序.

1. 输入呢?

用户点击按钮这个行为本身

2. 处理呢?

记录用户点击按钮的次数

3. 输出呢?

显示到屏幕上

下面是可以完成这个任务的程序.

以下代码为python版本.

```
1 import tkinter as tk
2
3 window = tk.Tk()
4 window.title('ClickLog')
5 window.geometry('300x300')
6
7 num = 0
8
9 def callback():
```

```

10 global num
11 num += 1
12 text.configure(text=f"这是你第{num}次点击按钮")
13
14 text = tk.Label(window, text="这里会显示你点击了几次按钮!", background="yellow",
15 fg="black", font=('Times', 15, 'bold'))
16 text.pack()
17
18 button = tk.Button(window, text="点击我!", command=callback)
19 button.pack()
20 window.mainloop()

```

3.1.3 无论什么编程语言,你都需要会他们

实际上,大部分的编程语言的语法都十分类似的,他们使用了相同的结构来组成这个语言的体系,所以理所当然的他们有着重合的部分.

3.1.3.1 注释

无论什么编程语言,注释都是最重要的部分之一.注释可以让你在程序中随意的书写你自己的想法而不会对程序产生影响.

而注释又分成了单行注释,多行注释,以及特殊的注释.

实际上,仅仅使用单行注释就已经满足了我们的正常使用.所以在任何接触到编程语言时我们都应该首先关注他的注释方式是什么.

在cpp中单行注释的方式是使用 \\\ 来表示注释.这意味着 \\ 后面的任何内容都不会被计算机当作语句来处理

在python中,单行注释的方式是使用 # 开头.

3.1.3.2 函数

任何编程语言都会有函数,这里的函数尽管抽象出来和数学上的函数是一样的,但是请不要当作一个简单的表达式去理解.

函数可以简化我们的代码,并提高我们代码的复用性.

为了解决一个问题,我们可能会多次用到同一个处理方法,那我们就不妨把他单独拿出来作为一个函数,这样我们每次再去写的时候就可以通过调用这个函数来避免多次撰写类似的语句了.

对比数学上的函数 $f(x) = 5x + 1$,我们发现他由输入值 x ,输出值 $f(x)$ 和解析式 $5x + 1$ 组成,所以函数也理所应当的由输入,处理方式,输出组成.

看到这里你可能会发现他和程序相同,是的,在我看来函数正是程序的微观体现.

下文会提到python的函数语法.

3.1.3.3 运算符

如同数学中的加减乘除运算一样,任何程序语言都需要有他们独特的运算符来表示运算.这个部分大部分语言都是类似的.

3.1.3.3.1 根据功能对运算符分类

1. 算数运算符

算数运算符就是我们理解的加减乘除,指数,幂运算等

2. 关系运算符

即是对两个对象进行比较.

也就是由大于(>),小于(<),大于等于(>=),小于等于(<=),相等(==),不相等(!=)组成.

3. 赋值运算符

对变量进行赋值操作的运算符.基本的是由简单的赋值操作符 = 构成的.

和算术运算符组合可以获得其他的赋值运算符,比如**加法赋值运算符**,也就是加等于(+=).

对于赋值操作符 =,他的表达式我有一点需要额外说明.

a=2 这个式子写作 a=2 读作 将2赋值给 a .也就是对于这个赋值操作符,请由右向左理解.

4. 逻辑运算符

逻辑运算符也就是与或非,可以简单理解为集合的交集,并集,补集等

5. 位运算符

位运算符是将变量直接当作二进制来进行处理的运算符,很容易理解,因为计算机底层就是使用二进制来保存的.

经典的位运算符也就是左右移动(>> 和 <<),按位与,按位或,按位非,按位异或.

tips:

1. 你可能会疑惑异或,非这些东西是什么这些东西就是一些简单的逻辑运算符,如果不理解可以去了解一点逻辑运算.
2. 位运算该如何去理解?这部分请去了解2进制和10进制之间转换的相关知识.
3. **请特别注意:一个等号 = 指的是赋值操作符,而不是判断是否相等!判断是否相等请使用两个等号 == .**

3.1.3.3.2 运算符的优先级问题

运算符具有优先级,就好像我们小学学过的一样,**先乘除再加减**.这意味着乘除的优先级比加减要高.每个语言都会有运算优先级表,但是实际上我们不需要背诵他,只需要有个印象就可以,因为我们可以使用括号改变优先级.

例如:

1	$5+6*7=47$
2	$(5+6)*7=77$

3.1.3.3.3 根据操作数对运算符进行分类

实际上运算符可以使用另一种分类方式,也就是按照操作数的数量进行分类.

1. 一元操作符

指的是操作数仅仅只有一个的操作符,最典型的就是 - ,这个符号并不是减号,而是负号,他的操作数只有一个.

再举例来说前置或者后置自增操作符(++)这个符号也是一元操作符,不过python中并没有这个操作符.

2. 二元操作符

大部分操作符都是二元操作符,二元操作符为对符号两边的表达式进行操作的运算符.

比如加号(+),就是对左右两边的表达式进行操作的符号. 5+6 就是对左右两个表达式 5 和 6 进行了求和操作.

同样的 (a+b)*(c-1) 这个式子中,乘号的左右两边的表达式分别为(a+b)和(c-1).

3. 三元操作符

三元操作符据我所知就一个,也即是三目运算符.

`(条件)?true:false` 这个运算符.他其实是一个简化的条件语句,下面会提到条件语句.不过python中也没有三目运算符.

三目运算符的优点在于可以简化代码结构,并进行简单的判断,比如奇偶数,单词的单复数等.

tips:

总的来说操作符和运算符的意思类似.为什么我在以上的部分的时候使用了操作符而不是运算符呢?

因为操作符可以是运算符,但是运算符不一定是操作符.

运算符一般认为是参与运算使用的操作符,比如数字计算,逻辑运算等等.

举个简单的例子,在python中定义一个数组 `a=[1, 2, 3, 4, 5]`.我们使用了 `a[0]` 来获取到数组中的第一个元素,也就是1.

这里的 `[]` 也是一个操作符叫做**下标访问操作符**,不过是一元操作符,他的操作对象是一个列表.

再举例而言,就是后文中3.4.2部分提到的函数为例.`filename.read()`.无论是这里的`.`还是`()`,他们都是一元操作符,这里的`.`叫做**成员访问操作符**,`()`则叫做**函数调用操作符**.成员访问操作符访问了对象(或者是变量)`filename`中的`read`方法.函数调用操作符则调用了`read`函数,并向其传递参数,不过这里没有参数被传递进函数.

3.1.3.3.4 概念由人定义

概念是被人定义的,所以从这个角度来说概念本身是没有错误的,比如我可以把街道上全是垃圾叫做赤潮,我很清楚赤潮并不会发生在街道上,但是如果我将街道上全是垃圾称作赤潮这个概念分享给了友人A(`bushit`),那么当我下次走到街道上的时候,当我对他说街上又发生赤潮了啊,他自然可以理解我的意思.

概念=概念的名称+概念内容.放到操作符上来说,**操作符=符号+行为**.操作符也是由人来定义的,这意味着什么?

这意味着两点.

1. 进行操作的操作符是可以自由定义的.(概念的名称)

此刻,我定义`~`这个操作符为`×`号,那么`8~9`自然就等于`72`.那么在这种情况下`×`就没有任何实际意义了.

2. 操作符对操作数进行的行为.(概念内容)

例如,我定义一个新的操作符`⊕`,并规定 $A \oplus B = A \times (A + B) + B \times (A - B)$.这个式子是随便写的,当然没有任何意义.但是我们可以像使用 $1 + 1 = 2$ 一样使用这个操作符 $2 \oplus 5 = 2$.

比较常用的是在对类对象进行操作的时候,对`-`操作符进行重载,防止没有使用我们自己设计的`-`的重载函数,编译器自动生成一个浅拷贝的赋值运算符的重载函数.

什么,你说上面一行字看不懂?那就当作没看到,只需要知道运算符的行为是可以被更改的就行了.

3.1.3.3.5 运算符参与化简代码

我们知道,`×`的出现,让我们书写 $2 + 2 + \dots + 2 = 2000$ (一共有 $1k$ 个 2)时不用写很长一段,我们只需要写 $2 \times 1000 = 2000$ 就可以了.它当然可以理解为 $1k$ 个 2 相加,也可以理解为 2 个 $1k$ 相加.

所以我们说`*`是`+`的简记法.

也就是说,运算符的出现是为了简化我们的运算.

比如`+=`这个符号,当我们需要写`a=a+b`时我们只需要写`a+=b`即可.这样就可以理解编程中很多莫名其妙的运算符,他们本质上来说就是为了简化我们的代码.

毕竟我们肯定更希望使用`**`来参与计算而不是使用`pow(x, y)`函数来参与计算,这一点也不直观.

再举一个运算符化简代码更加明显的例子.

获取二进制的输入.

```
1 if (a == 0)
2 {
3     printf("输入合法!");
4 }
5 else if (a == 1)
6 {
7     printf("输入合法!");
8 }
9 else
10 {
11     printf("输入非法!");
12 }
```

我们知道,对于二进制而言,可以接受的数字无非0和1,所以只要输入是0或者1即可.

```
1 if (a == 0 || a == 1)
2 {
3     printf("输入合法!");
4 }
5 else
6 {
7     printf("输入非法!");
8 }
```

我们看到上面的代码就比第一个代码更简单.它使用了逻辑运算符或(||).

以下代码为python版本.

```
1 if(a==0):
2     print("输入合法!")
3 elif(a==1):
4     print("输入合法!")
5 else:
6     print("输入非法!")
```

```
1 if(a==0 or a==1):
2     print("输入合法!")
3 else:
4     print("输入非法!")
```

3.1.3.4 数据类型

如同整数可以分为正整数负整数,以及0这样的分类一样,计算机中储存的数据也可以被分成各种各样的类别.

常用的类型有以下几种

1. 整数(int)
2. 布尔(bool)
3. 浮点数(float)
4. 双浮点数(double)
5. 字符(char)

6. 字符串(string)

以上的内容可以进行扩展进而有无符号整型(unsigned int也就是包含0和正整数),有符号整型(signed int)也就是负数正数零.

短整型(short int),长整型(long int)等等

当然了还有更加复杂的类型比如python中的字典(Dictionary),列表(List)等.

请注意,数据类型所代表的数字范围并不是无穷大的,比如int可以表示的数字范围是-2147483648 到 2147483647.

原因也很好理解,int所占的字节数为4个字节,而一个字节是8比特,所以int可以表示的数字范围最大应该是 $2^{(32-1)} - 1 = 2147483647$.

来解释一下这个式子.

1. 为什么底数是2?

一个比特位有0和1两种情况

2. 为什么指数是32 - 1?

因为最高位需要作为符号位,不然不能表示负数.

3. 为什么要减1?

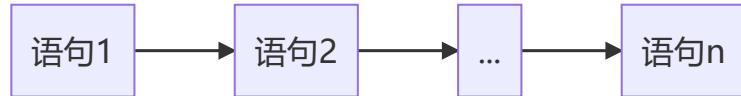
因为有一个数字0.

最小想必会算了吧?

3.1.3.5 语句

3.1.3.5.1 普通语句

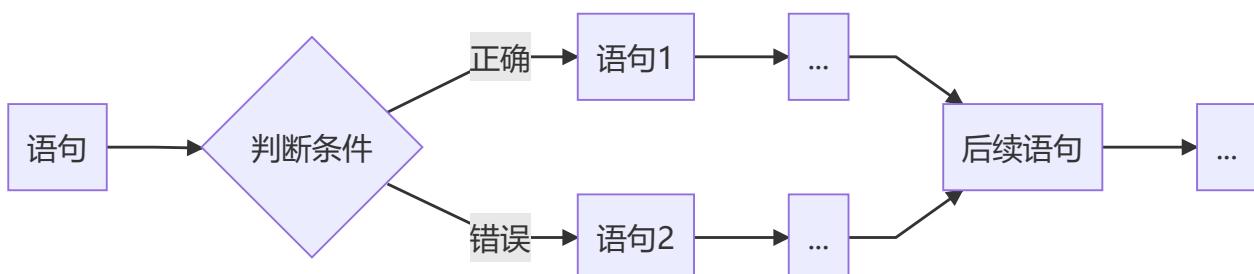
这个就是最正常的程序结构,程序由上到下依次执行,他不会有什么多余的事情,也没有什么好讲的.



1	语句1
2	语句2
3	...
4	语句n

3.1.3.5.2 条件语句

条件语句会改变程序执行的顺序.



经典的条件语句一般有两种

1. 单分支条件语句

```
1 | 语句
2 | if (判断条件)
3 | {
4 |     语句1
5 |     ...
6 | }
7 | else
8 | {
9 |     语句2
10|    ...
11| }
12| 后续语句
13| ...
```

2. 多分支语句

下面这个是数值匹配语句,即判断数值和数值几匹配,就执行第几个语句组,都不匹配就执行默认(default).

```
1 | 语句
2 | switch (判断数值)
3 | {
4 | case 数值1:
5 |     语句1
6 |     ...
7 |     break;
8 | case 数值2:
9 |     语句2
10|    ...
11|    break;
12| ...
13| case 数值n:
14|     语句n
15|     ...
16|     break;
17| default:
18|     语句n+1
19|     break;
20| }
21| 后续语句
```

下面这个是python中经典的多分支,实际上就是使用单分支进行了嵌套.

```
1 | 语句
2 | if 判断条件1:
3 |     语句1
4 |     ...
5 | elif 判断条件2:
6 |     语句2
7 |     ...
8 | ...
9 | elif 判断条件n:
10|     语句n
11|     ...
12| else:
```

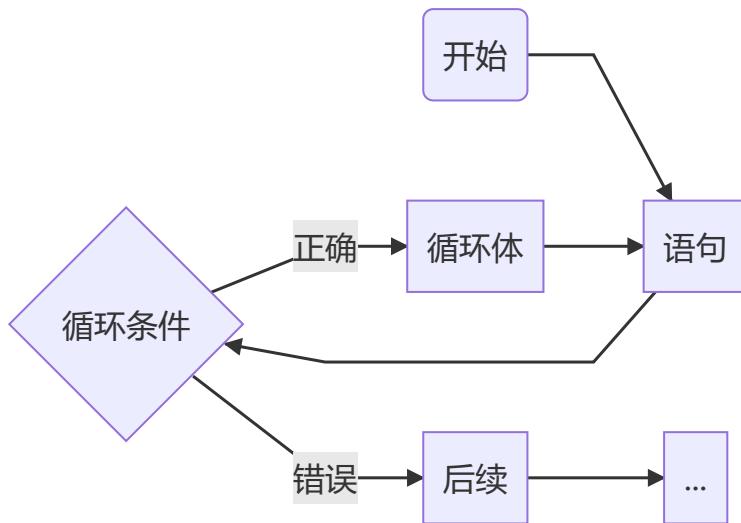
13 | 语句n+1
14 | ...
15 | 后续语句

`elif`也就是`else if`的缩写,所以这个其实就是在不满足第一条件的前提下进行了多次判断的语句组合.

值得一提的是,默认执行语句并不是必须的.也就是说无论是第一个例子中的`default`还是第二个例子中的`else`都是可选的.

程序在没有匹配到对应的数值,即没有满足的条件下,会执行默认语句,如果没有默认语句,那当然什么都不会执行了.

3.1.3.5.3 循环语句



如上结构,程序由**开始**节点开始执行程序,判断是否符合**循环条件**,如果符合**循环条件**,就一直执行**循环体**内容,直到不满足**循环条件**,结束循环,执行**后续代码**

在上一句话中,我提到了**结构**,而不是**语句**,是因为为了达到**循环**的效果,实际上并不是只有**循环语句**才可以做到,还有比如**递归**.但是**循环**和**递归**有着本质的区别,这里我暂且不讨论**递归**.

为了保证**循环**的有效性,所以**判断条件**是必不可少的,程序中一般会尽量避免出现**死循环**也就是程序会一直**循环**并不会**结束运行**.

但是仍然有例外,也就是我在 3.1.2.2 中的那个例子.

3.1.2.2 中是一个窗口带有按钮,为了检测到用户是否按下了按钮,这个循环会在用户使用这个软件的过程中一直不断的执行,这样才可以保证无论用户什么时候按下鼠标点击按钮,都可以立刻作出反应.毫无疑问,如果你在使用软件的过程中,如果每一步操作都需要等待一定时间才会有反馈,那么使用体验是绝对非常不好的.这个循环也被称为是**主循环**.

循环一般有以下几种.

1. for 循环

```
1 for(定义变量;判断条件;改变变量)
2 {
3     循环体
4 }
5 //如下是一个具体的例子
6 for(int i = 0;i <= 5; i++)
7 {
8     cout<<i;
9 }
```

2. while循环

```
1 while(条件)
2 {
3     循环体
4 }
```

`while` 也是最常用于主循环的语句,如下语句可以很简单的写一个主循环.

```
1 while(true)
2 {
3     cout<<"Loading..."<<endl;
4 }
```

这样这个窗口上就会一直输出 `Loading...` 字样.

3. dowhile循环

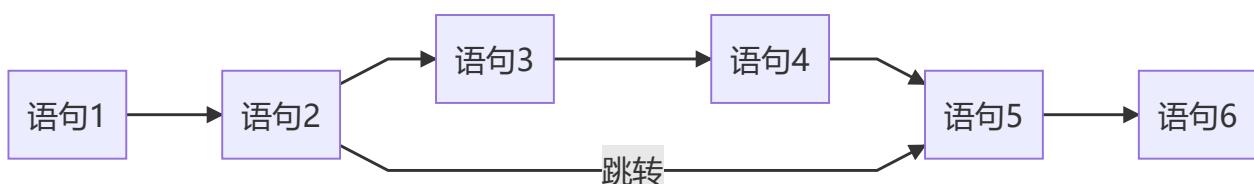
```
1 do
2 {
3     循环体
4 }while(判断条件);
```

`do...while` 循环的和 `while` 循环的不同之处在于, `do...while` 循环会在判断条件之前,先执行一遍循环体的内容.

以上的三个循环并不是python中的语法.但是在大部分编程语言中,都会使用这三种循环,在做出修改.

3.1.3.5.4 跳转语句

这个语句并不常用,一般来说会结合条件语句使用,单独使用的情况基本没有.



如上图所示,由于语句2是跳转语句,所以执行到语句2的时候就会直接跳转到语句5部分,相当于语句3和语句4部分白写了,所以一般才会配合条件语句.

```
1 语句1  
2 语句2  
3     goto 标记5;  
4 语句3  
5 语句4  
6 标记5:  
7 语句5  
8 语句6
```

我们可以观察到,语句34直接被跳过.

举个具体的例子

```
1 tag:  
2     cout<<"Tag"<<endl;  
3     goto tag;  
4 cout<<"finish"<<endl;
```

我们观察到 `finish` 永远不会被输出,因为他会在执行这句话之前跳转到 `tag` 处.

也就是说,这也是一种循环的方式.因为没有人规定跳转的目标位置只能在哪里,所以它可以在任意的地方.

关于3.1部分的内容,尤其是3.1.3处的内容

1. 首先可能会看的云里雾里,这是正常现象,当你学习了多门语言之后,看起来会轻松很多,尽管 我将它写在这里,我是希望你有一个大概的印象,当你阅读完3.2部分,可以回过头来再看一遍在这个地方,相信会有更好的理解.
2. 关于3.1.3中给出的所有例子

这里给出的所有例子仅仅表达思想,并不具体指哪门编程语言,尽管可以和cpp对应.

这里所有的语句组都可以和流程图对应,在阅读对应代码的时候还请对应相应的流程图来阅读.

顺便一提,绘画流程图是一个好习惯,在编写复杂的程序之前,绘制一份流程图以清晰自己该做什么是一个好选择.

好的,接下来可以讲一讲python的基础语法了.

3.2 Python的基础语法

虽然说是基础语法,但是实际上我并不会一一告诉你语法,因为这些东西的详细内容你可以通过看书和文档来解决.

这意味着,这部分你需要对应着文档和教程对照着看.也许你会在下面发现和文档不同的情况,此时,请不要相信我写的内容,也不要轻信文档,请注意,程序就在你的手边,实践往往是检验真理的唯一标准.所以动手试试就知道了.

关于文档和结果不统一的情况,一般有两个原因.

1. 文档并不是官方的,也就是没有及时更新到最新版本此时请访问官方的文档以确保正确性.
2. 官方并未及时更新文档或者你发现了一个官方写在文档中的错误.

恭喜你,你可以反馈给官方看.

3. 程序版本与文档版本并不符合.

说明书都拿错了,谈何对错呢?

3.2.1 关于Python不得不说的事

3.2.1.1 打开Python idle

在阅读下面的这些内容之前,还请打开python的idle,这样你就可以动手跟着做了.

在你的安装目录下找到 python.exe 执行程序,执行他,可以打开一个黑色的窗口,如下

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

同样的,你也可以在开始菜单的所有应用选项中找到idle这样一个选项,打开的会是这样一个窗口

The screenshot shows the IDLE Shell 3.10.2 window. The title bar reads "IDLE Shell 3.10.2". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python 3.10.2 startup message:

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
```

The command prompt ">>> |" is visible at the bottom left. The status bar at the bottom right shows "Ln: 3 Col: 0".

这两个窗口的效果都是一样的,你可以在这里面输入python的指令,他会直接执行很有趣,不是吗?

还有其他的方法,比如使用 `win+r` 输入 `cmd` 打开命令提示符界面.

```
Microsoft Windows [版本 10.0.22631.3880]
(c) Microsoft Corporation。保留所有权利。
C:\Users\LENOVO>
```

当你得到这个界面之后,你可以输入python也同样可以进入python的idle界面.

```
C:\Users\LENOVO>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

当你在命令行提示符中看到了三个右尖括号 >>>,这意味着你已经进入了python idle模式.

这里我更加建议你使用系统自带的命令行进入Python idle模式下,因为我们可以顺便检查一下版本.

详细的内容我将会在终端部分进行详细的解读,这里就先略过这个陌生的窗口吧.

tips:

下文中所有的python代码你都可以在这个idle窗口中进行输入并观察效果.

但是记得cpp的代码就不可以了.

3.2.1.2 Python的语言特性

Python是一门解释型的交互语言.

3.2.1.2.1 什么是解释型?

首先我们知道,计算机底层是由0和1组成的,所以实际上我们所写的语句如同 `HelloWorld` 这样的话计算机并不认识.所以我们所写出来的程序计算机是不认识的,这里就要经过一个名为**编译**的步骤.

编译,就是把程序变成计算机可以看得懂的样子,然后我们才可以执行它.

举个不恰当的例子,我们将蔬菜清洗,处理好这部分就是我们在写代码,然后放在锅里烹饪,这部分就是编译,然后我们可以吃了这道菜,这就是执行.尽管这个例子不够具体也不够严谨,但是我想为了可以吃(为了可以让计算机执行)这样一个目的而进行烹饪(编译)的这样一个思想我应该有传达到.

也许最恰当的例子就是翻译官了吧,当我们和一个英国人沟通交流的时候,我们都不会说对方的语言,于是找了一个翻译官,翻译官在**保证意思不变的前提下**,将中文转化成为了英文,说给了英国人听.这就是编译过程.这里翻译的目的是为了让英国人听得懂,达到沟通交流的目的.

如果我需要举一个反例,那么我想cpp就是最典型的例子,你无法直接双击 `*.cpp` 结尾的文件,你不会看到他成功运行,你需要将他转化为 `*.exe` 才可以观察到程序的运行效果.也就是所谓的**编译并运行**.

但是如果是python文件,即 `*.py` 却可以双击直接运行.确实我们会看到黑色的命令提示符窗口,但是他确实仅仅通过双击就运行起来了不是吗?

当然了,你会发现,如果你仅仅拷贝 `*.py` 文件到一台新电脑上,是运行不起来的,这里就还请看3.3部分来了解详细情况吧.

3.2.1.2.2 什么是交互型?

还记得我们打开了一个idle窗口吗?我们可以在上面一行一行的输入代码,就好像我们在使用终端一样.

这就意味着这门语言是实时交互的,因为我们在命令行中输入命令并按下回车,我们可以立刻看到效果.

试着在 `>>>` 后输入 `print("HelloWorld!")` 并执行.我们立刻发现idle输出了 `HelloWorld!` 在这句命令下面,这就是我们在和idle进行实时交互的最好证据.

3.2.2 Python与众不同的缩进

与cpp不同,python并没有使用`;`来作为一行语句的结束,而是使用了换行和缩进来改变程序的结构.

就像句号代表一句话说完一样,在cpp中,分号代表了一句话说完了.

```
1 | int a = 0; //有分号,说明这句话说完了
2 | int b = 1 //没有分号,程序认为这句话没说完,所以程序会执行错误
```

但是在python中,你的每句话都不需要使用分号来结束.在输入新的语句的时候,你仅仅只需要换行即可.

这也意味着cpp可以把多行语句写在同一行,而python不行.

```
1 | int a = 0;int b = 1;
```

这句话不会报错,尽管我们平时并不这么写,因为代码并不整洁.

而python写这句话只能老老实实的如下写

```
1 | a=0
2 | b=1
```

当然了,python也提供了简化的方案

```
1 | a,b=0,1
```

同样的,cpp也可以简化

```
1 | int a = 0,b = 1;
```

我们在中文中常常使用段落来表示一个部分,就如同程序中的一个个部分一样,而在cpp中,我们一般使用{}来表示一个部分.如下.

```
1 | if (true)
2 | {
3 |     //部分1
4 |     cout<<"true";
5 | }
6 | else
7 | {
8 |     //部分2
9 |     cout<<"false";
10| }
```

我们观察到上面有两组{},也即是说上面的代码被分成了两个部分.

这意味着我们在部分1中无论写多少句话,只要仍然在花括号之间,就仍然属于部分1.部分2同理.

在python中,我们使用了缩进来表示部分,也就是层级.

```
1 | if a==0:
2 |     printNum(n)
3 | else:
4 |     print(a,b)
```

我们观察到,python中使用了:来作为各个部分的开始,而使用了一个缩进来区分每个部分.

也许这样看起来似乎很简单,但是实际上在实际上开发过程中,会有相当多的部分结合在一起.

tips:

我们来试一试idle吧!

在idle中输入a=1这句话,按下回车键.

你可能会以为什么都没有发生,因为idle并没有显示什么,其实idle已经将a=1这个数值存储起来了.

试着输入a并按下回车键试试看吧!

3.2.3 Python注释

1. 单行注释

以#号为开头即可

2. 多行注释

多行注释之间的部分书写注释无需使用单行注释#开头

1. 三个单引号'''

2. 三个双引号 """

```
1 # 这是单行注释
2 """
3 这是多行注释
4 多行注释之间的部分书写注释无需使用单行注释#开头
5 注释2
6 注释3
7
8
9 注释n
10 """
11 """
12 """
13 这是另一种多行注释的方法
14
15
16
17
18 空白...
19
20
21 你可以写任何你想要写的东西,反正他不会干扰程序的正常执行
22 """
```

3.2.4 函数

3.2.4.1 函数的定义

```
1 def 函数名(参数列表):
2     函数体
3     返回值
```

有些地方的教程上面会把返回值并到函数体中,这也行,因为**函数不一定需要返回值**.

```
1 def functionName(Num1, Num2):
2     print(Num1, Num2)
```

以上的这个例子就没有返回值.

3.2.4.2 函数的调用

函数的调用是由**函数名(参数列表)**组成的调用,下面是一个简单的函数调用的例子.

```
1 def functionName(Num1, Num2):
2     print(Num1, Num2)
3     return
4 functionName(5, 6)
```

运行上面的程序,你就会看到终端输出了5 6这两个数字.

3.2.5 运算符

除去常规的加减乘除的常规运算符号之外,我特别提到以下三个运算符需要注意.

1. 取模运算符 %

求取除法的余数,最常使用的场景是判断奇数还是偶数.

```
1 | if Num%2==0:  
2 |     print("偶数")  
3 | else:  
4 |     print("奇数")
```

2. 幂运算符 **

返回的是x的y次方

```
1 | a =1 + 10 ** 5
```

这个就是科学计数法的简单表示,即一乘以十的五次方.

3. 取整除运算符 //

取整除运算符就是我们写算式中的商.

```
1 | 9 ÷ 2 = 4……1
```

我们看到上述算式中余数就是取模运算符返回的值,而商就是取整除运算符返回的值.

这里有一个特别要提醒的点,基于不同的版本,python中**除法有不同的表现**.

1. 在2.x版本中,使用整型相除得到的结果是一个整型.

也就是说使用 `1/2` 这样的式子返回的是 `0`.其实很好理解,1除以2的商正是0;

如果我们需要返回0.5,那我们需要使用浮点数相除,也就是 `1.0/2.0`.他会返回 `0.5`.这也正是数据类型存在的意义.

2. 但是3.x的版本中,哪怕你是用的是整数,返回的也会是一个浮点数(小数).

关系运算符不讲,赋值运算符不讲,逻辑运算符不讲.

额,逻辑简单提一嘴吧.

逻辑最简单的是与或非.

1. 与,也就是和(AND),可以理解为交集.

也就是说同时满足了两个条件才可以,可以理解为电路上串联了两个开关,只有两个开关都闭合,电路才会连通.也就是都是开这个状态的重合部分.

2. 或(OR),可以理解为并集

就是说满足任意一个条件即可,可以立即为并联两个开关,无论哪个接通了都可以让电路联通.

3. 非,可以理解为补集

举例来说,整数中非0的数正是整数中对0取补集的部分,也就是正整数和负整数.

位运算符提一个左右移动运算符.

左右移动运算符指的是将二进制下的数字进行左右移动,满足**高位丢弃,低位补0**原则.

比方说十进制5的二进制数是 0000 0101 ,那么 $5 \ll 3$ 的结果就是40,二进制是 0010 1000 ,很容易观察到二进制向左边移动了两位.

$5 \ll 3$ 等价于 5×2^3 ,那么公式为 $x \ll y == x * 2^y$.

此处我使用了数学中的表达方式,请在计算机中依照计算机的表达方式.

1. 乘号(*)使用星号(*)来表示,比如 $5 * 6$
2. 次方使用 \wedge 来表示,比如二的三次方为 2^3
3. 除号(\div)使用/来表示,如 $5 / 6$

以后的内容我将使用符合计算机的表达方式.

python中其他的运算符还请自行查阅文档学习.

3.2.6 数据类型

3.2.6.1 如何知道一个数数据的数据类型?

使用一元操作符 `type()`.

在python IDLE中尝试输入一个变量 `a=5` ,按下回车后再输入 `type(a)` .观察输出.

The screenshot shows the Python IDLE Shell interface. The title bar reads "IDLE Shell 3.10.2". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the Python interpreter's prompt and some sample code:

```
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> a=1.1
>>> type(a)
<class 'float'>
>>> a=5
>>> type(a)
<class 'int'>
>>>
```

The output shows that `a=1.1` is of type `float` and `a=5` is of type `int`. The status bar at the bottom right indicates "Ln: 9 Col: 0".

我们观察到当a等于5的时候a的数据类型为 int ,而当a等于1.1的时候,a的数据类型为 float .

tips:

这里我使用了"当a等于5的时候"这句话,是因为这句话我仅仅指的是 type(a) 这个前提下.

如果我以 a=1.1 和 type(a) 两个命令为前提时,我应该说 将1.1赋值给a之后,使用a的数据类型为float .

当然了,其实不论怎么说,我想人们都是听得懂的.

3.2.6.2 关于数据类型,我们应该关注什么?

1. 有哪些数据类型?
2. 不同的数据类型之间怎么转化?

在python中定义一个变量的方式是使用赋值操作符.以 名字=值 的方式进行定义.

变量就好像是一个杯子,杯子里放多少水,放什么水都是可以决定的.多少水,指的就是值的变化,而什么水,指的就是值的类型.

相比没有人会使用玻璃来承装氢氟酸吧?这正是因为类型不同,就好像我们试图使用 int 类型来承装 string 类型一样.

不过幸运的是,python定义变量的时候并不需要我们写明类型,因为python会自动读取类型并赋值.

广义上来说,数据类型只有三种.

1. 数字类型

可以是整数(int),也可以是浮点数(小数, float, double)等等.

2. 字符类型

可以是单个字符(char),也可以是字符串(string).

字符串也就是由多个字符组成的有序列表.

3. 布尔类型

也就是真(TRUE)和假(FALSE).不过也可以讲真假当作0和1处理,这样布尔类型就可以并入数字类型中.

3.2.7 语句

终于来到了语句部分.学到这里,你就已经具备写出简单的程序的能力了,不过你可能会觉得写不出东西,这很正常,因为你需要训练.

四种语句在这里我不讲普通语句和跳转语句.我只讲条件和循环.

3.2.7.1 条件语句

在 3.1.3.5.2 部分我提到了python中的条件语句,我拿来放在这里.

```
1 语句
2 if 判断条件1:
3     语句1
4     ...
5 elif 判断条件2:
6     语句2
7     ...
8 ...
9 elif 判断条件n:
10    语句n
```

```
11     ...
12 else:
13     语句n+1
14     ...
15 后续语句
```

这里我想详细讨论的是判断条件部分,判断条件应该是一个**返回值为布尔值的表达式**.

这意味着判断条件可以是 `6>5`,这和 `true` 等价.

我还提到了 `switch` 语句,在python中, `switch` 语句被替换了 `match...case` 语句,但是结构大差不差.

3.2.7.1 循环语句

python中的循环语法部分还请自行学习,这里我想要就 `for` 循环的结构来讲一讲 `range()` 函数.

```
1 for 变量 in 列表:
2     循环体
```

列表可以是 `range()` 产生的列表,也可以是自己手动写出来的一个列表变量,也可以是某个不知名库中的某个函数所返回的列表.

总的来说,只要是列表即可.

```
1 range(开始,结束,步长)
```

对于`range`产生的列表,我们可以简单理解为**输出开始+步长*次数直到这个值大于等于结束**.

值得注意的是这里的次数由0开始,所以输出的第一个数永远是**开始这个值**.

1. 关于开始这个参数,他的默认值为零,这意味着我们不写他,程序会默认由0开始.
2. 结束这个参数没什么好说的.
3. 步长这个参数默认值为1.

结合以上三个参数的规则,我们不难发现 `range(10)` 和 `range(0,10,1)` 完全等价,由于 `开始` 和 `步长` 的默认值就分别是0和1,所以设置他们没有必要.

又因为 `开始` 和 `步长` 都有默认参数,所以当我们仅仅只写了一个参数的时候,自然要给没有默认参数的 `结束` 了.

综上, `range()` 可以产生一个左闭右开的列表.

3.3 写完程序该做什么?

在3.2.1.2.1中我提到如果你将你的程序(`*.py`文件)拷贝到其他新的电脑上去,你会发现运行不了,这是为什么呢?

这里有两个重点,分别是 `*.py` 文件和新的电脑.为什么一定是这两个条件呢?

新的电脑保证的是这台电脑没有安装python,而 `*.py` 保证的是你拷贝的是在python环境下运行的代码本身而不是程序.

也就是说, `*.py` 文件之所以可以成功运行并出现效果(例如打开了一个窗口)是因为有python环境的存在,如果没有python环境,那么 `*.py` 和 `*.txt` 文件没有任何区别.

你可以想象一个歌手,他使用一个带有变声器的麦克风,,唱歌当然很好听可是如果换了其他没有变声器的麦克风呢?自然就不堪入耳了.

所以这个歌手需要训练,直到有一天,他不需要变声器也可以唱的很好听,这样他无论使用的是什么麦克风,是否使用麦克风,就都不会影响他唱的歌曲了.

这就是我们在写完程序后该做的事情,让他可以在任意一台电脑上运行.

为了在任意一台电脑上运行,我们需要将 *.py 文件进行打包,将他转化成为 *.exe 文件.

将 *.py 文件进行打包的方法有很多,这里我介绍一种,使用第三方模块库 pyinstaller.

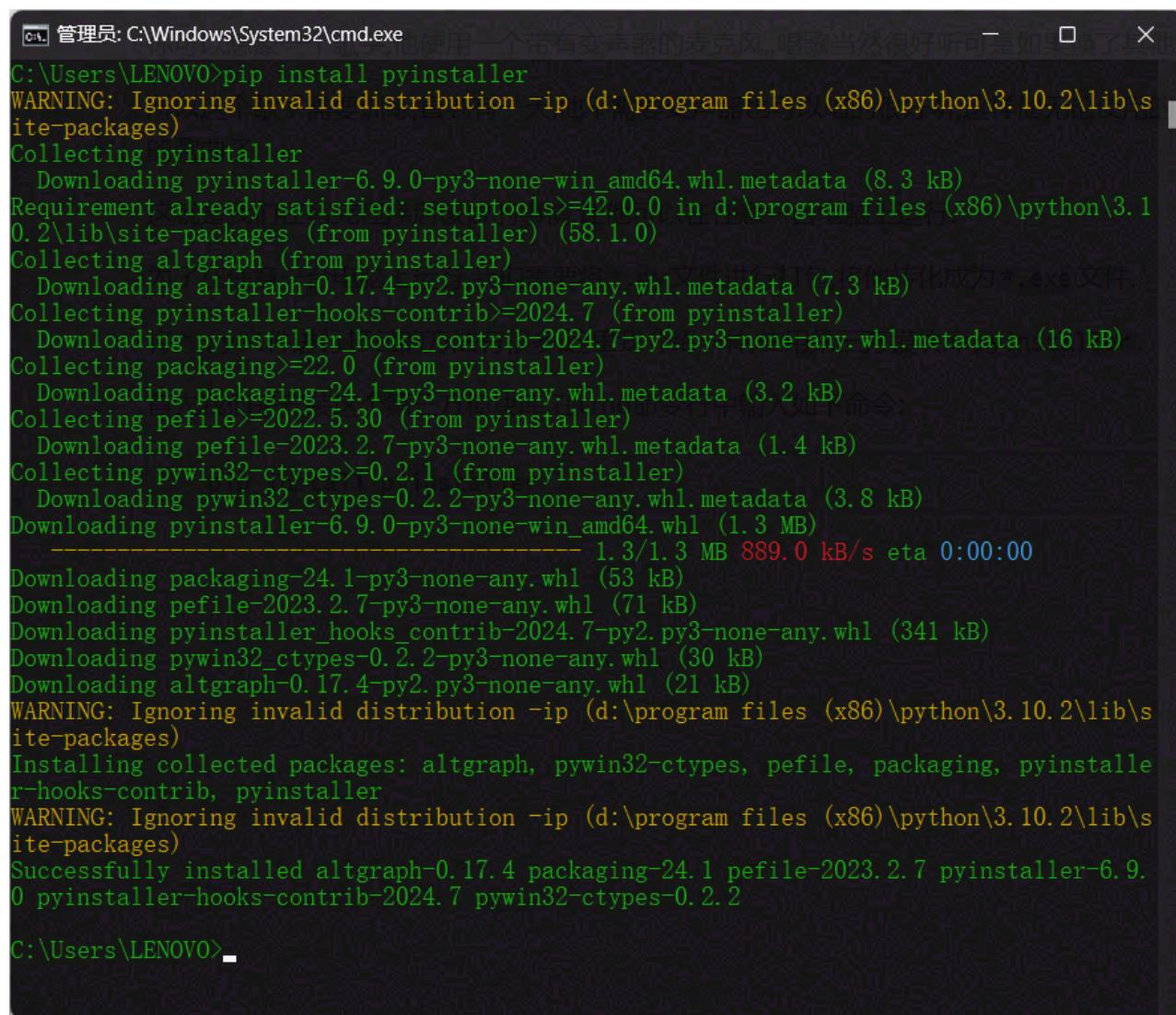
3.3.1 安装 pyinstaller 模块

首先我们来安装这个第三方模块库.在cmd命令行中输入如下命令:

```
1 | pip install pyinstaller
```

如果pip使用失败可以尝试使用 pip3,因为在python3后python自带的pip版本更新至3.

如果没有意外,那么你的界面也许会是这个样子的.



```
C:\Users\LENOVO>pip install pyinstaller
WARNING: Ignoring invalid distribution -ip (d:\program files (x86)\python\3.10.2\lib\site-packages)
Collecting pyinstaller
  Downloading pyinstaller-6.9.0-py3-none-win_amd64.whl.metadata (8.3 kB)
Requirement already satisfied: setuptools>=42.0.0 in d:\program files (x86)\python\3.10.2\lib\site-packages (from pyinstaller) (58.1.0)
Collecting altgraph (from pyinstaller)
  Downloading altgraph-0.17.4-py2.py3-none-any.whl.metadata (7.3 kB)
Collecting pyinstaller-hooks-contrib>=2024.7 (from pyinstaller)
  Downloading pyinstaller_hooks_contrib-2024.7-py2.py3-none-any.whl.metadata (16 kB)
Collecting packaging>=22.0 (from pyinstaller)
  Downloading packaging-24.1-py3-none-any.whl.metadata (3.2 kB)
Collecting pefile>=2022.5.30 (from pyinstaller)
  Downloading pefile-2023.2.7-py3-none-any.whl.metadata (1.4 kB)
Collecting pywin32-ctypes>=0.2.1 (from pyinstaller)
  Downloading pywin32_ctypes-0.2.2-py3-none-any.whl.metadata (3.8 kB)
  Downloading pyinstaller-6.9.0-py3-none-win_amd64.whl (1.3 MB)
----- 1.3/1.3 MB 889.0 kB/s eta 0:00:00
Downloading packaging-24.1-py3-none-any.whl (53 kB)
Downloading pefile-2023.2.7-py3-none-any.whl (71 kB)
Downloading pyinstaller_hooks_contrib-2024.7-py2.py3-none-any.whl (341 kB)
Downloading pywin32_ctypes-0.2.2-py3-none-any.whl (30 kB)
Downloading altgraph-0.17.4-py2.py3-none-any.whl (21 kB)
WARNING: Ignoring invalid distribution -ip (d:\program files (x86)\python\3.10.2\lib\site-packages)
Installing collected packages: altgraph, pywin32-ctypes, pefile, packaging, pyinstaller-hooks-contrib, pyinstaller
WARNING: Ignoring invalid distribution -ip (d:\program files (x86)\python\3.10.2\lib\site-packages)
Successfully installed altgraph-0.17.4 packaging-24.1 pefile-2023.2.7 pyinstaller-6.9.0 pyinstaller-hooks-contrib-2024.7 pywin32-ctypes-0.2.2
C:\Users\LENOVO>
```

接下来输入下面这个命令来确定是否成功安装了 pyinstaller 模块.

```
1 | pip list|find "pyinstaller"
```

并使用如下命令来查看详细信息.

```
1 | pip show pyinstaller
```

```
C:\Users\LENOVO>pip list|find "pyinstaller"
WARNING: Ignoring invalid distribution -ip (d:\program files (x86)\python\3.10.2\lib\site-packages)
pyinstaller          6.9.0
pyinstaller-hooks-contrib 2024.7

C:\Users\LENOVO>pip show pyinstaller
WARNING: Ignoring invalid distribution -ip (d:\program files (x86)\python\3.10.2\lib\site-packages)
Name: pyinstaller
Version: 6.9.0
Summary: PyInstaller bundles a Python application and all its dependencies into a single package.
Home-page: https://www.pyinstaller.org/
Author: Hartmut Goebel, Giovanni Bajo, David Vierra, David Cortesi, Martin Zibricky
Author-email:
License: GPLv2-or-later with a special exception which allows to use PyInstaller to build and distribute non-free programs (including commercial ones)
Location: d:\program files (x86)\python\3.10.2\lib\site-packages
Requires: altgraph, packaging, pefile, pyinstaller-hooks-contrib, pywin32-ctypes, setuptools
Required-by:

C:\Users\LENOVO>
```

在他的详细信息界面,你可以看到他的名字,版本,概要,主页,作者,许可证等等等等非常多信息.

这里出现的三行命令我都会进行详细介绍. pip 是python的一个下载工具,我会在后文介绍,其余的命令部分我会在终端部分进行详细的介绍.

接下来输入 `pyinstaller --version` 来查看模块的版本.值得注意的是,查看版本往往成为确定是否成功安装的一种方法.

理论上版本应该是6.9.0.

3.3.2 开始第一次打包吧

这里我仅仅介绍最简单的打包方法,不过这也足够使用一段时间了,当你遇到了问题,也就是打包出现了问题的时候,还请自行查询相关信息解决问题.

打包的前提是你已经有了一个 `*.py` 文件需要打包,那么我们可以在当前目录下打开cmd窗口,或者是将我们cmd窗口的地址更改到有这个 `*.py` 文件的地方去,至于为什么,我会在终端那一节告诉你.

3.3.2.1 最简单的打包方法

在对应目录的文件夹下按住 `shift` 并按下鼠标右键,打开右键菜单.

你应该会看到下面这个菜单.

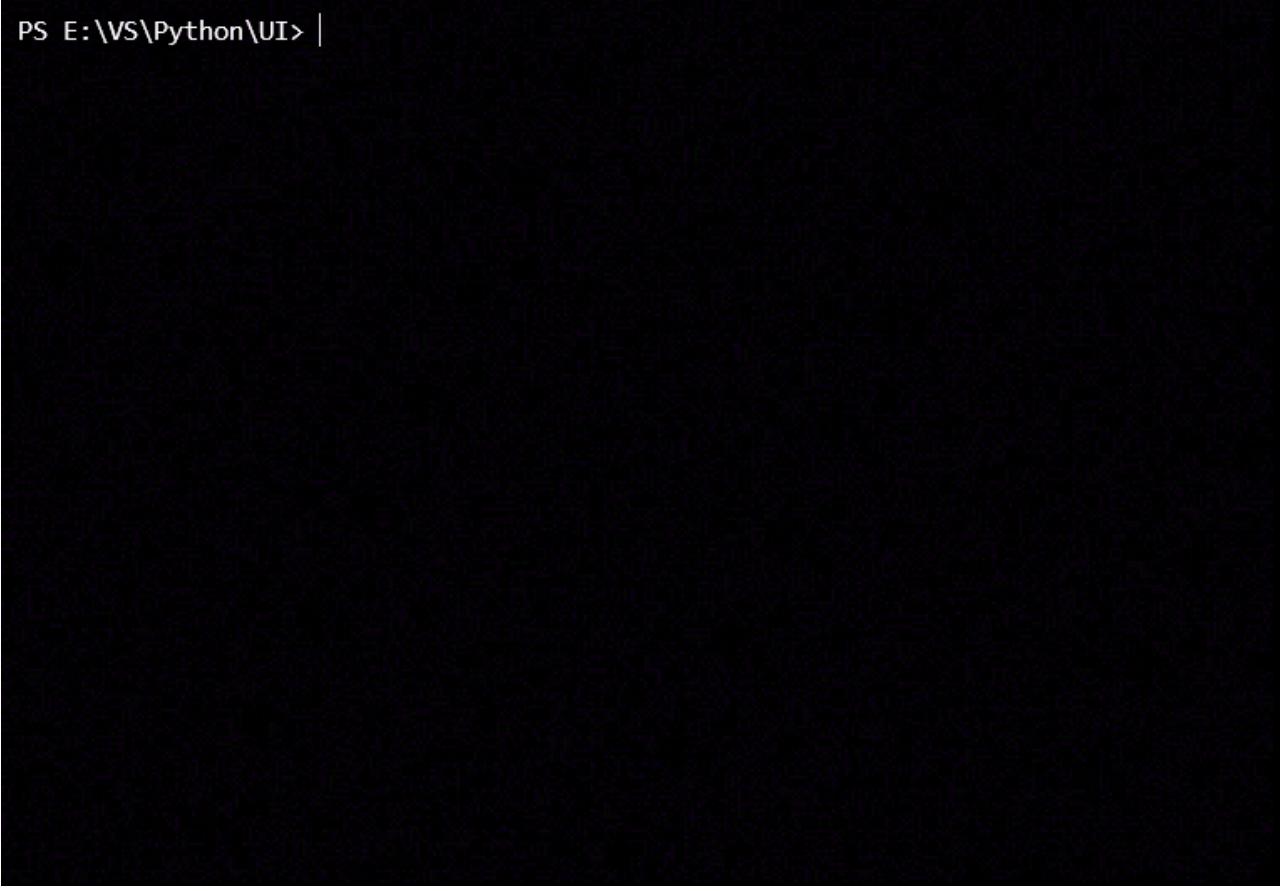


也许我们的右键菜单不一样,原因是**win11使用了新版菜单,而我将菜单改回了win10旧版**.尽管如此,你依然可以通过**更多**这样的选项打开新版的详细菜单,还请关注这两个选项.

- 在终端中打开
- 在此处打开Powershell窗口

这里我们使用第二个,因为有些人(比如我)会更改默认终端的设置,在这里我的**在终端中打开**选项并不能满足我的要求.

但是使用第二个是可以满足我的要求的.



PS E:\VS\Python\UI> |

你可以看到他打开了一个黑色的窗口.

你应该会发现这个菜单中的文件路径,在我这里是 E:\vs\Python\UI ,和你的 *.py 文件的存储路径是一样的,或者说,就是你 shift+右键 的文件夹位置.

我们可以使用 dir 或者 ls 命令来查看当前目录下的文件.

```

PS E:\VS\Python\UI> pwd
Path
-----
E:\VS\Python\UI

PS E:\VS\Python\UI> ls

    目录: E:\VS\Python\UI

Mode LastWriteTime Length Name
---- ----- ----- ----
-a--- 2024/7/31   47556 a.exe
-a--- 2024/8/1    311 print.cpp
-a--- 2024/8/1    48544 print.exe
-a--- 2024/8/1    349 printNum.py
-a--- 2024/7/31   465 test.py

PS E:\VS\Python\UI> |

```

我们可以看到目录下是有我们要打包的这个文件的(`test.py`).

对了,旁边的这个日期指的是最后一次保存的日期.虽然说是最后一次编辑的日期,其实就是保存啦.

那么当我们目录下已经确定有了我们需要打包的文件我们的打包就可以开始了.

还记得我们`test.py`文件内部的代码吗?他就是我们在 [3.1.2.2](#) 中所写的代码.如果已经不记得了还请跳转回去看看,不过也并不是让你去看这个代码,而是回忆他的效果.

输入 `pyinstaller` 文件名可以进行简单的打包,还记得 `range()` 函数吗?这里和他类似,缺省的参数全部使用了默认的值,而打包哪个文件则是如同**结束**这个参数一样,不可以缺省的.很好理解,你观察到我上图中除了一个名叫 `test.py` 的 python 文件之外,还有 `printNum.py` 的 python 文件和 `print.cpp` 的 cpp 文件.如果不告诉 `pyinstaller` 你需要打包哪个程序,它又怎么知道该打包哪个呢?

那么打包 `test.py` 这个文件的最简单命令就是 `pyinstaller test.py`,不过还请不要打包,因为打包出来的结果有两种,如下.

- 文件夹模式

打包结果是一个文件夹,文件夹中有一个exe文件,用来运行程序.

打个比方,桌面上的 qq 程序,你可以右键并**打开文件所在位置**,你很容易就会发现这个文件是一个文件夹中的一个exe的一个链接.你双击这个exe文件,就可以启动qq,不过如果单独把QQ.exe这个文件拷贝到桌面上,那么你双击的时候并不会成功运行程序,说明这个程序使用了文件夹模式打包.

当你想要传递这个软件的时候,你只需要将这个文件夹压缩并发送给需要的人,他们就可以很轻松的双击运行这个程序.

当然了,复杂的大型程序还是不能这样做的,毕竟你安装qq的时候是使用了一个安装程序,而不是解压了一压缩包.不是吗?

◦ 优点

1. 便于调试,你可以通过文件夹下的文件看出来有哪些模块被打包了.
2. 便于更新,由于模块和主程序并没有打包到一起,如果只更改了主程序,并没有添加新的模块,我们可以简单的替换前后的exe文件达到更新的效果

尽管我并不认为这样的更新方式是正确的,因为你永远不会理解用户会如何使用你的程序,以及他们会在一个简简单单的复制替换过程中做出多少额外的事情导致程序不能正常运行,但是他确实方便.

3. 速度快.在打包前后,文件的运行速度会受到影响,但是如果使用了文件夹模式打包,那么它的运行速度和打包前基本不变.

◦ 缺点

文件体积会变大,因为许多文件并没有被压缩.

• 单文件模式

仅有一个exe文件组成,民间俗称绿色模式.

原因是非常方便,常见的例子就是各种民间小软件了,如修改注册表的小插件,管理资源管理器的小插件 `MyComputerManager-x64.exe`,他们都是绿色模式进行的打包.

尽管我叫他们小插件,但是实际上他们仍然是软件,并不是插件.

◦ 优点

1. 便于传递,上面我也说到了,原因是非常方便,所以才会受民间喜爱.他太过于方便了,下载一个exe文件双击就直接运行了,这对于任何一个电脑小白来说都是无可避免的福音.
2. 体积小.既然文件夹模式体积会变大,那么单文件模式下体积变小也不难猜测,原因当然是很多文件被压缩了.这样也更可以体现了方便这个特点,毕竟民间的小软件都是为了实现某个单一功能而现的,所以体积小,自然便于传递了.

总的来说,绿色模式的文件最大的优点就似乎方便.

◦ 缺点

1. 打开变慢啦!打包成为单一的exe文件之后文件的打开速度会变慢,不过由于民间的小软件体量小,所以其实这个问题并不是什么问题,但是对于稍大型的软件来说,打开时间变长就会极大程度的影响使用体验了,毕竟你也不想打开一个qq等待1分钟吧?

2. 附加文件变得麻烦

你可以理解为不好携带说明书.大部分人并不会把说明书放到程序中,因为这只会添加不必要的工作,但是绿色版软件的用户群体又有很大一部分小白,如果没有说明书,可能他们根本不会用,更不要说民间写出来的小软件质量参差不齐,有的设计软件UI的时候根本不走心,觉得毕竟只是个小软件,没有必要认真设计的人也是存在的.

对于说明书文件,一般指的是 `README.md` 的文件.我会在后面的部分为你简单介绍这个东西,现在我们先关注于 `pyinstaller` 的使用吧.

3.3.2.2 更改参数来控制打包

那么我们该如何控制使用哪种模式来打包呢?这里就要引入参数的概念了.

还记得我们之间查看 `pyinstaller` 版本的命令吗?

```
1 | pyinstaller --version
```

这里的 `--version` 就是参数.

由此我们可以知道,参数是改变命令的设置,达到更加精细程度的控制的选项.

参数一般有两种.

- 由一个 - 引领的参数.

比如查看 python 的版本,在命令行中的命令就是 `python -v`.

- 由两个 - 引领的参数

比如上文中查看 `pyinstaller` 版本的命令.

由此我们可以得出以下结论.

1. 参数的结构是由 - 引领,后跟随参数的名称.

2. 单个 - 引领的参数是简写版本,比如 `python -v` 中的 `-v` 等价于 `--version`.这里的 v 就是 version 的缩写.

不过并没有大小写的要求,并不是说简写参数就一定要大写.比如 `pyinstaller` 中 `--version` 参数的简写就是 `-v`

3. 两个 - 引领的参数是全称版本.比如 `pyinstaller --version`.

请注意以上参数的规则,这很重要,在后面的终端部分我们还会反复用到上述结论.因为参数的规则同样适用于终端.

下面我将介绍常用的参数列表.

名称	描述
<code>-D</code>	文件夹模式(默认值)
<code>-F</code>	单文件模式
<code>-c</code>	打包后运行时显示控制台窗口(默认值)
<code>-w</code>	打包后运行时隐藏控制台窗口
<code>-i [image path]</code>	设置软件的图标
<code>-h</code>	显示帮助信息
<code>-n [file name]</code>	更改文件名
<code>--clean</code>	在构建之前清理pyinstaller缓存并删除临时文件

这几个参数就是我们最常使用的了.

我们现在来输出一个默认参数下的包,体会一下打包的过程.

```
1 | pyinstaller text.py
```

如果你的文件名并不是 `text.py` 还请记得更改.

```

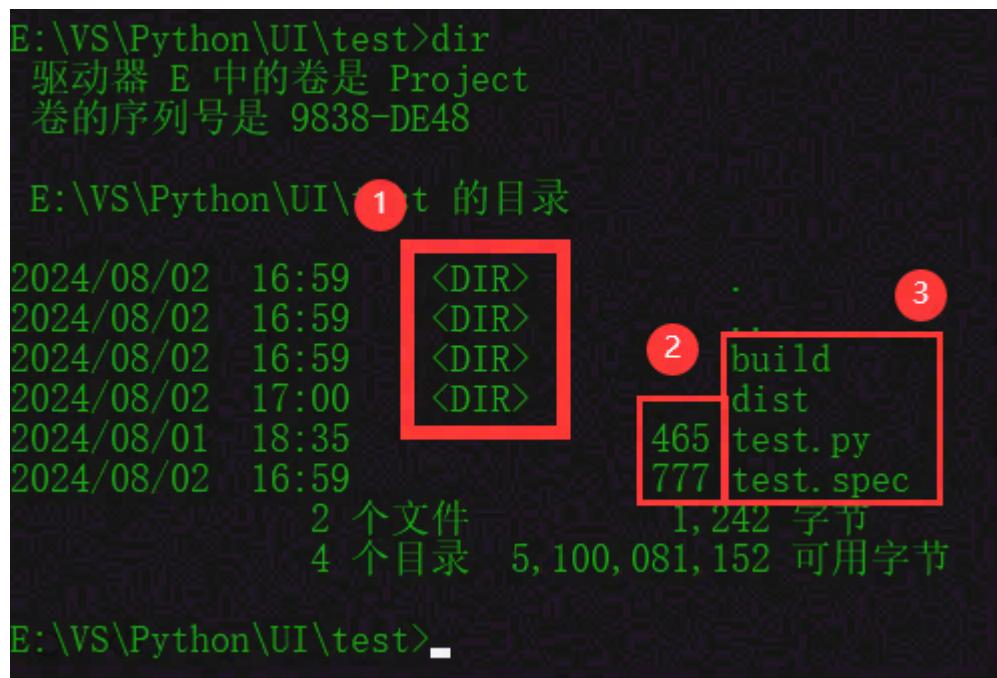
管理员: C:\Windows\System32\cmd.exe
10370 INFO: Loading module hook 'hook-pickle.py' from 'D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\hooks'...
13494 INFO: Caching module dependency graph...
13810 INFO: Looking for Python shared library...
13856 INFO: Using Python shared library: D:\Program Files (x86)\Python\3.10.2\python310.dll
13857 INFO: Processing E:\VS\Python\UI\test\test.py
1428 INFO: Loading module hook 'hook-_tkinter.py' from 'D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\hooks'...
1445 INFO: Processing module hooks...
1460 INFO: Loading module hook 'hook-_tkinter.py' from 'D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\hooks'...
1464 INFO: checking Tree
1466 INFO: Building Tree because Tree-00.toc is non existent
1466 INFO: Building Tree Tree-00.toc
1466 INFO: checking Tree
14662 INFO: Building Tree because Tree-01.toc is non existent
14662 INFO: Building Tree Tree-01.toc
14723 INFO: checking Tree
14724 INFO: Building Tree because Tree-02.toc is non existent
14725 INFO: Building Tree Tree-02.toc
14776 INFO: Performing binary vs. data reclassification (924 entries)
16644 INFO: Looking for ctype DLLs
16653 INFO: Analyzing run-time hooks...
16654 INFO: Including run-time hook 'D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\hooks\rthooks\pyi_rth_inspect.py'
16655 INFO: Including run-time hook 'D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\hooks\rthooks\pyi_rth_tkinter.py'
16759 INFO: Looking for dynamic libraries
17353 INFO: Extra DLL search directories: [AddDllDirectory: []]
17353 INFO: Extra DLL search directories (PATH: [])
1440 INFO: Warnings written to E:\VS\Python\UI\test\build\test\warn-test.txt
15505 INFO: Graph cross-reference written to E:\VS\Python\UI\test\build\test\xref-test.html
16672 INFO: checking PVZ
16673 INFO: Building PVZ because PVZ-00.toc is non existent
16674 INFO: Building PVZ (ZlibArchive) E:\VS\Python\UI\test\build\test\PVZ-00.pvz
20324 INFO: Building PVZ (ZlibArchive) E:\VS\Python\UI\test\build\test\PVZ-00.pvz completed successfully.
20381 INFO: checking PKG
20391 INFO: Building PKG because PKG-00.toc is non existent
20391 INFO: Building PKG (Archive) test.pkg
20443 INFO: Building PKG (Archive) test.pkg completed successfully.
20449 INFO: Bootloader D:\Program Files (x86)\Python\3.10.2\lib\site-packages\PyInstaller\bootloader\Windows-64bit-intel\run.exe
20451 INFO: checking EXE
20452 INFO: Building EXE because EXE-00.toc is non existent
20453 INFO: Building EXE from EXE-00.toc
20456 INFO: Copying bootloader EXE to E:\VS\Python\UI\test\build\test\test.exe
20501 INFO: Copying icon to EXE
20533 INFO: Copying O resources to EXE
20534 INFO: Embedding manifest in EXE
20561 INFO: Appending PKG archive to EXE
20569 INFO: Fixing EXE headers
205818 INFO: Building EXE from EXE-00.toc completed successfully.
20594 INFO: checking COLLECT
20594 INFO: Building COLLECT because COLLECT-00.toc is non existent
20902 INFO: Building COLLECT COLLECT-00.toc
20920 INFO: Building COLLECT COLLECT-00.toc completed successfully.

E:\VS\Python\UI\test>

```

执行完程序后你会看到超级长一串 `INFO`,请等到命令执行结束,也就是上图中最后一个 `INFO: Building COLLECT COLLECT-00.toc completed successfully`.这代表打包成功了.

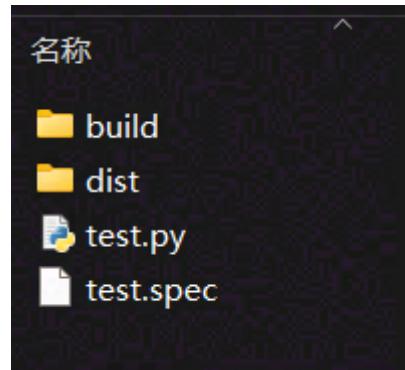
输入 `dir` 命令可以看到当前文件夹下的文件.



1. 如果带有 `<DIR>` 标识,代表这个文件是一个文件夹.
2. 如果这是一个文件,这里显示的就是这个文件的大小,单位是字节,最下方是总的大小.
3. 这部分是文件夹或者文件的名字.

考虑到这里还没有学习终端相关的知识,所以我决定还是使用图形化UI来讲解,这样也许会更加容易立即.

打开我们刚刚打包的那个文件夹.



我们可以看到多出了两个文件夹和一个文件.

- build文件夹

包含了日志等信息

- dist文件夹

内部就是打包的结果

- test.spec文件

这个是打包的参数配置文件,如果你需要重复的打包而并不更改参数,你可以不使用 `pyinstaller [一大堆参数] test.py` 这样的命令来打包,可以简单的使用 `pyinstaller test.spec` 来打包.

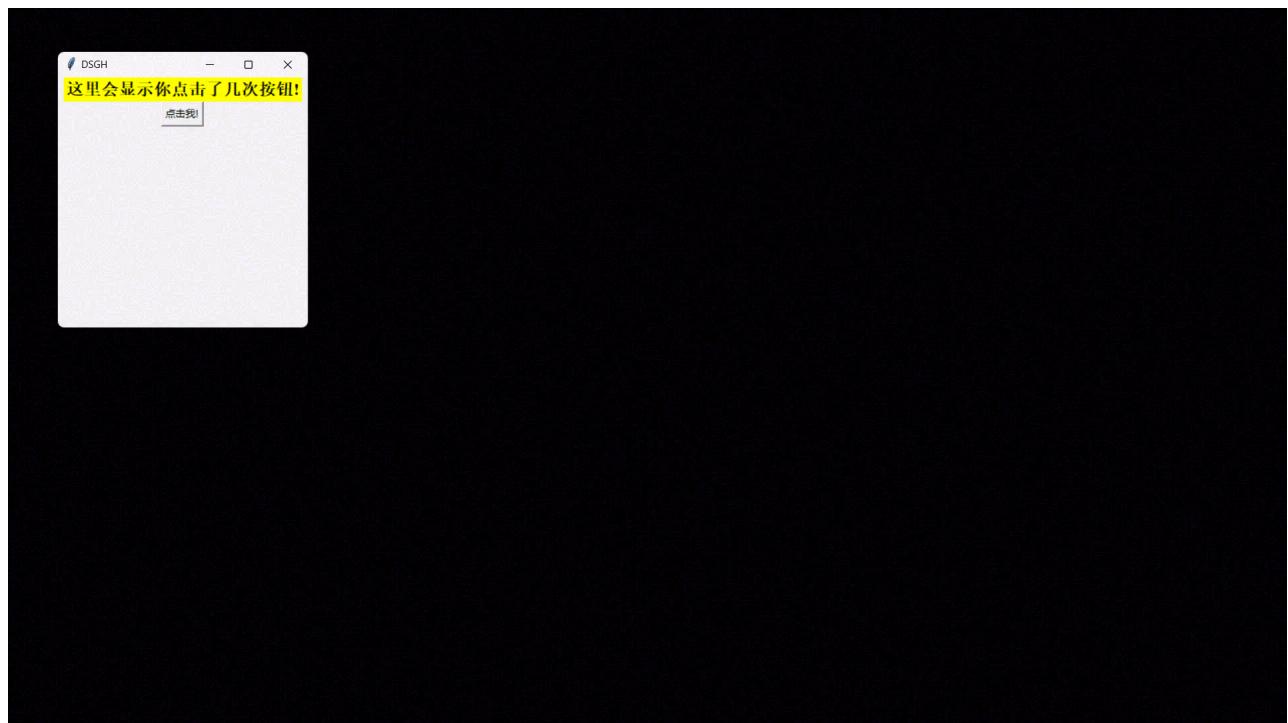
这意味着如果我们可以编辑 `*.spec` 文件,就可以实现无需传入一大堆参数的打包,是的,这部分是个稍微进阶的内容,但是并不困难,我只会在后文简单介绍使用方法,详细的使用方法建议自学.

简单提一嘴,生成 spec 文件需要使用到一个叫做 `pyi-makespec` 的工具.不过你并不需要额外的安装他,因为他已经包含在了 `pyinstaller` 内部.

如果你已经对这个文件非常熟悉,你甚至可以直接使用记事本来打开他,效果是一样的.

好的,了解到了打包后的各个文件和文件夹是做什么的后,我们来打开我们的打包结果,看一看有什么是我们不满意的.

进入 `dist` 文件夹,并找到 `test.exe`,双击打开,结果如下.



我们观察到他打开了一个命令行,显然它并不美观,也没有实际作用,所以我们将它隐藏起来.在参数列表中我提到了可以使用 -w 参数来隐藏这个控制台.



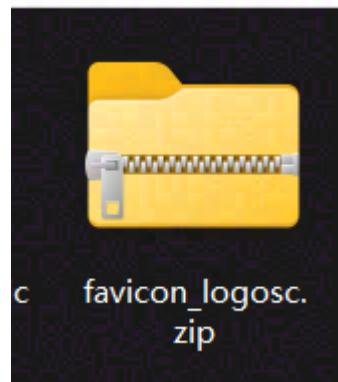
这是 dist/test 文件夹下的文件,我们发现程序的入口, test.exe 文件似乎并不符合我们的需求,他的图标并不好看,以及他的名字也并不是我们需要的.以及这个软件仅仅只有一个记录点击了几次按钮的功能,所以并不需要使用文件夹模式来打包.

使用 -F 参数来改变输出模式为单文件模式.

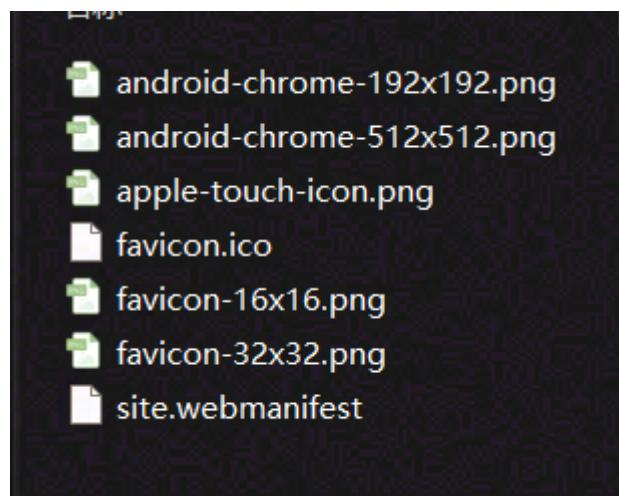
然后我们来更改他的图标.默认图标并不好看,我们需要使用 -i 参数来设置他的图标.

-i 参数需要传递进入一个图片,这个图片的类型应该是 *.ico .我们可以使用 Photoshop 来制作一张 *.ico 的文件.不过使用美术软件生成好看的图标并不是我们的任务,所以我们可以使用简单的图片转换 *.ico 功能即可.

点击[这里](#)打开一个在线的ico图标制作网站,你可以在上面下载现有的,也可以上传图片生成自己想要的.因为这个小软件是记录点击按钮的次数,所以我就使用我的光标指针图像作为软件的图标了.



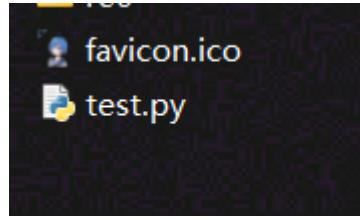
这是下载的结果,一个压缩包.



这是解压后的内容,他把所有基本需要使用的格式全部生成了一份.

其中我们需要关注的就是 favicon.ico 这个文件.至于为什么叫做这个名字,这和 html 设置网页图标有关系,我们先不管他.

现在我们就可以来尝试打包了.将 favicon.ico 文件复制到和 test.py 文件同一目录下.



现在在这个目录下执行打包命令.

```
1 | pyinstaller -F -i ./favicon.ico -w test.py
```

生成结束进入新的 dist 目录,可以看到打包的结果.



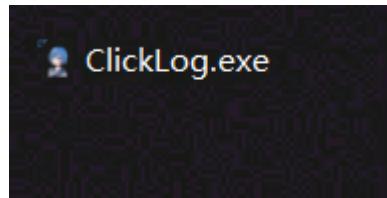
一个已经更改了图标的软件.

有的人并不喜欢使用在线的网站来制作 *.ico 文件,这确实麻烦,如果可以在命令行解决一切,那自然最好,我们可以使用 pillow 模块.

后文会有详细的介绍.这里暂且先使用在线网站制作 *.ico 文件吧

添加 --clean 参数来在构建之前清理pyinstaller缓存并删除临时文件.目的是防止上一次的打包结果对这次的打包产生影响.

使用 -n [filename] 参数来控制输出文件的文件名,我们将它的文件名更改为 ClickLog .



这是我们生成的结果,我们双击打开发现打开速度较慢,如果没有鼠标指针的变化提醒我们软件已经打开了,那么我想很多人可能会反复点击软件来确保打开了软件,所以我们需要给他们一个软件已经打开了的提示,避免他们反复打开软件.

为了使用 splash 功能,我们需要在代码前面添加两句.

```
1 | import pyi_splash  
2 | pyi_splash.close()
```

这两句话是用于引入 pyi_splash 模块,并将这个模块创建的闪屏界面关闭,如果不将他关闭的话,那么这个闪屏界面会一直存在,直到我们的应用程序关闭为止.

python可能会提醒 pyi_splash 模块找不到,不要鸟他,程序可以正常打包执行,但是程序无法被python直接执行,如果想要在这种情况下使用python执行,需要改用下面的代码:

```
1 | try:  
2 |     import pyi_splash  
3 |     pyi_splash.close()  
4 | except ImportError:  
5 |     pass
```

这里使用了 `try-except` 命令,让python发现导入 `pyi_splash` 失败,然后就跳过了这两句话,所以程序才可以正常运行.

`try-except` 是专门用来处理异常的语句,他也是一个分支语句我在 `except` 部分使用了 `ImportError` 来指定异常类型是导入 `pyi_splash` 模块失败时执行下边的代码.(因为我们从报错知道python无法正常导入这个模块.)

部分报错信息:

```
1 | ModuleNotFoundError: No module named 'pyi_splash'
```

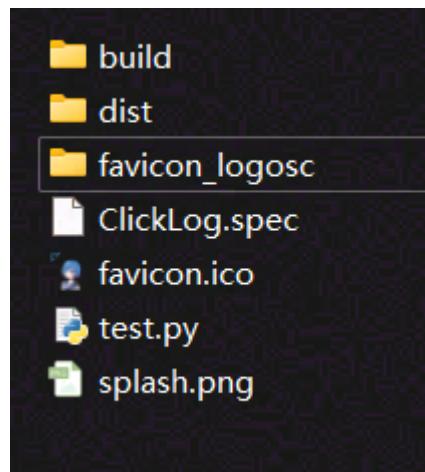
如果导入失败,那么就是用 `pass` 语句跳过上面的语句.

如果不清楚不关闭 `splash` 有什么后果,可以暂时把这句话注释掉,尝试打包程序并运行,查看后果是什么.

我随便找了一张图片作为提示,并放在 `test.py` 同级目录下,命名为 `splash.png`.请注意,这里的文件只能是 `*.png` 格式.并且尺寸还有限制,所以我们实际上并不会直接使用原始图片打包.

如何转换格式?

还请不要直接更改后缀,对于新手来说,转换文件格式的最简单的办法就是下载软件[格式工厂](#),上手难度几乎为0,操作也简单易懂.



这是现在目录下的文件.

- `build` 和 `dist` 文件夹, `ClickLog.spec` 文件是上一次的生成结果.
- `favicon_logosc` 文件夹是使用[标小智](#)生成的图标文件夹.
- `test.py` 是我们的程序本体.
- `favicon.ico` 是来自 `favicon_logosc` 文件夹的 `ico` 文件
- `splash.png` 是我们的设置的启动画面文件.

介绍完以上内容,我们开始打包.

```
1 | ValueError: ('The splash image dimensions (w: %d, h: %d) exceed max_img_size (w: %d, h:%d), but the image cannot be resized due to missing PIL.Image! Either install the Pillow package, adjust the max_img_size, or use an image of compatible dimensions.', 1080, 608, 760, 480)
```

执行打包结束后,如果有这样的输出提示,说明图片的尺寸并不符合比例.

不过我们也不需要去裁剪等等,因为我们现在将使用 `pillow` 这个模块.

使用 `pip` 安装 `pillow` 模块,命令如下.

```
1 | pip install pillow
```

确认安装成功之后,我们就可以使用 `pillow` 这个模块了.那么她有什么优点呢?

- 设置 `ico` 图标的时候,不限制格式, `pillow` 会自动设置
- 使用 `splash` 闪屏的时候, `pillow` 会自动更改所有的参数,同样不限制格式.
- `pillow` 和 `pyinstaller` 配合度很高,你并不需要额外的配置,只需要安装好了 `pillow` 这个模块,发现 `ico` 文件的格式并不正确, `pyinstaller` 会自动调用 `pillow` 模块做出修改.

```
1 | pyinstaller -Fw -i .\favicon_logos\favicon-16x16.png --clean -n "ClickLog" --splash  
.\\splash.png test.py
```



这是输出结果,我们可以看到双击文件的时候,先是出现了一张图片,随后图片关闭,程序才打开.

3.3.2.3 优化打包命令和使用spec文件

观察我们的打包命令.

```
1 | pyinstaller -Fw -i .\favicon_logos\favicon-16x16.png --clean -n "ClickLog" --splash  
.\\splash.png test.py
```

是否觉得他真的太长了呢?我们发现 `-Fw` 是将 `-F` `-w` 参数进行了合并,这样达到了一定程度上的化简.但是这种化简方式还是有很多问题,首先有参数的 `-i` 和 `-n` 无法合并.全称参数 `--clean` 和 `--splash` 也无法合并.

这里就要使用我们上文中提到的 `spec` 文件了.

我们可以简单的使用上一次生成的 `spec` 文件来进行生成,请注意,因为使用了上一次生成的 `spec`,所以参数自然和上一次一样.

```
1 | pyinstaller ClickLog.spec
```

我们可以看到结果和上一次相同.程序也可以正常运行.

```
1 | pyi-makespec -Fw -n "ClickLog" -i .\favicon_logos\android-chrome-192x192.png --  
splash .\\splash.png test.py
```

使用这条命令来创建一个 `spec` 文件.

如果成功运行,结果应该如下.

```
E:\VS\Python\UI\test>pyi-makespec -Fw -n "ClickLog" -i .\favicon_logos\android-chrome-192x192.png --splash .\\splash.png test.py  
Wrote E:\VS\Python\UI\test\ClickLog.spec.  
Now run pyinstaller.py to build the executable.
```

值得注意的是,我们发现上面的那条命令并没有 `--clean` 参数,这个参数是不可以包含在 `pyi-makespec` 命令中的,因为如果使用了 `spec` 的话,我们需要额外在 `pyinstaller` 中指定 `--clean` 命令.

```
1 | pyinstaller --clean ClickLog.spec
```

也许你会发现删除 --clean 参数程序依然可以正常的生成,但是实际上会发现有影响,而且这个问题在中文搜索下有且仅有一篇[文章](#)提到了解决方法但是由于标题问题又不会被需要解决这个问题的人搜索发现,大部分人可能遇到了这个问题都会一头雾水不知如何解决,所以不妨加上这个参数,基本上就不会出现第一次的程序影响第二次的程序的问题,他也并不会增加多少打包的成本,何乐而不为?

执行可以正确生成文件.

关于spec的内容就介绍到此.

3.3.2.4 所有代码提供

1. test.py 内容

```
1 try:
2     import pyi_splash
3     pyi_splash.close()
4 except ImportError:
5     pass
6
7 import tkinter as tk
8
9 window = tk.Tk()
10 window.title('DSGH')
11 window.geometry('300x300')
12
13 num = 0
14
15 def callback():
16     global num
17     num += 1
18     text.configure(text=f"这是你第{num}次点击按钮")
19
20 text = tk.Label(window, text="这里会显示你点击了几次按钮!", background="yellow",
21 fg="black", font=('Times', 15, 'bold'))
22 text.pack()
23
24 button = tk.Button(window, text="点击我!", command=callback)
25 button.pack()
26
27 window.mainloop()
```

2. shell命令

```
1 pyi-makespec -Fw -n "ClickLog" -i .\favicon_logos\android-chrome-192x192.png -
-splash .\splash.png test.py
```

3. spec文件内部内容

```
1 # -*- mode: python ; coding: utf-8 -*-
2
3
4 a = Analysis(
5     ['test.py'],
6     pathex=[],
7     binaries=[],
```

```

8     datas=[],
9     hiddenimports=[],
10    hookspath=[],
11    hooksconfig={},
12    runtime_hooks=[],
13    excludes=[],
14    noarchive=False,
15    optimize=0,
16 )
17 pyz = PYZ(a.pure)
18 splash = Splash(
19     '.\\splash.png',
20     binaries=a.binaries,
21     datas=a.datas,
22     text_pos=None,
23     text_size=12,
24     minify_script=True,
25     always_on_top=True,
26 )
27
28 exe = EXE(
29     pyz,
30     a.scripts,
31     a.binaries,
32     a.datas,
33     splash,
34     splash.binaries,
35     [],
36     name='clickLog',
37     debug=False,
38     bootloader_ignore_signals=False,
39     strip=False,
40     upx=True,
41     upx_exclude=[],
42     runtime_tmpdir=None,
43     console=True,
44     disable_windowed_traceback=False,
45     argv_emulation=False,
46     target_arch=None,
47     codesign_identity=None,
48     entitlements_file=None,
49     icon=['favicon_logos\\favicon-16x16.png'],
50 )

```

3.3.3 分享与使用

当你打包好一个程序之后,你就可以在 `dist` 目录下拷贝出软甲,进行分享和使用了.

- 如果是文件夹模式打包请将整个 `dist` 目录下的所有文件进行压缩,然后分享,不要只分享 `exe` 文件!
- 如果是单文件模式打包,分享的时候建议附加说明书(`README.md`),而不是仅仅一个单文件应用程序.

3.4 python不那么基础的基础

3.4.1 pip详细介绍

我们使用了pip工具安装了 `pyinstaller` 和 `pillow` 两个模块,可以说我们已经在一定程度上非常熟悉这个工具了.

在python3以上的版本,pip自动包含在了python中,他已经是python不可分割的一部分.我们来介绍pip.

3.4.1.1 常用二级命令

名称	用途
install	安装包
uninstall	卸载包
list	已安装的包的列表
freeze	按照规定的方式显示已安装的包的列表
check	检查安装的包是否按成功安装了相应的依赖项
show	展示已安装包的信息
search	搜索包的信息

什么是二级命令?别找了,好像没有这个术语.

二级命令指的是在主命令下的分命令,例如

```
1 | pip install [packagename]
```

这里pip是主命令,告诉电脑,我们使用pip这个程序,随后install就是二级命令它来告诉pip我使用了这个命令,随后的[packagename]指的是传入的install的参数,用来告诉pip安装的是什么包.

常见的还有 `net user`,这个命令中 `net` 是主命令,而 `user` 是次命令,也是二级命令.

以下每个二级命令都有复数个参数,但是我只会提到常用的参数.

参数同样分为一级参数和二级参数,理论参考二级命令.一级参数对二级命令生效,二级参数仅可对该二级命令生效,无法影响其他二级命令.

1. install命令

```
1 | pip install [packagename]
```

在install后加上需要安装的包的名字,比如 `pip install pyinstaller`.

常用参数:

- `-U`:升级(更新)包.

升级pip:

```
1 | pip install --upgrade pip
```

2. uninstall命令

```
1 | pip uninstall [packagename]
```

和install命令类似.

3. list命令

显示已经安装的所有包的信息..

常用参数:

- -v 查看所有包的版本和安装地址.
- -o 显示所有可以升级的包.

4. freeze命令

简化版本的list命令.直接敲就可以看到包的版本,不需要加 -v 参数.

5. check命令

```
1 | pip check [packagename]
```

检查包是否安装成功了依赖项.

依赖项:

如同制作一辆汽车一样,我们不可能从炼铁开始,最多少买回来轮胎,引擎等物品,在工厂中组装.

程序同样是一样的,有的功能别人已经开发过了,我们无需从零开始开发,直接使用别人写好的工具即可,别人写好的这就叫做依赖项.如果没有了这些依赖项,那么很明显,程序就无法正常运行,就好像没有零件的汽车厂无法生产汽车一样.同样的,也叫做轮子,会在模块部分再次提到轮子这个概念.

6. show命令

```
1 | pip show [packagename]
```

显示包的详细信息.

7. search命令

```
1 | pip search [packagename]
```

搜索对应包

3.4.1.2 更改镜像源

如果发现pip下载速度很慢,可以来设置这个,如果并不慢,就不要更改了(我就没有更改).

pip下载网址默认在国外,可以使用国内的清华大学的镜像网站.地址是:<https://pypi.tuna.tsinghua.edu.cn/simple>(不是打开来用的)

1. 临时使用指令

```
1 | pip install -i https://pypi.tuna.tsinghua.edu.cn/simple [packagename]
```

2. 更改默认设置

```
1 | pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

3.4.2 文件的处理

3.4.2.1 输入和输出

除了以下这两个函数还有别的函数,但是还请自行学习.

1. 输入函数 `input()` 读取一行的输入内容
2. 输出函数 `print()` 输出一行内容

3.4.2.2 打开关闭文件

文件打开了就必须要有关闭.

1. 打开文件 `open(filename, mode)` .

`mode`常用参数列表.

参数名	效果
r	只读模式,光标默认在文件开头(默认模式)
r+	读写模式,光标默认在文件开头
w	只写模式,从头开始写入,也叫做覆写模式.
w+	读写模式,从头开始写入.
a	写入模式,在文件的末尾追加内容,也叫做追加模式
a+	读写模式,在文件的末尾追加内容
b	二进制模式打开

使用一个对象承接打开的文件,方便用于处理.

```
1 | f=open('./out.txt', 'wb+')
```

这里有一个有意思的问题,是否使用二进制打开有什么区别? 还请自行查阅.

提示:隐式转换,windows系统和linux系统的区别,换行符.

2. 关闭文件 `close()` .

没有什么好说的,在处理完文件后记得关闭文件即可.

3.4.2.3 读取写入文件

1. 读取函数read系列

将读取到的文件当作一个列表,里面每一行都是列表的一个元素.

1. `read([size])`函数

读取给定的字节数,不写(缺省)则读取全部.

2. `readline([size])`函数

读取单独的一行.可以理解为读取列表的一个元素.

请注意,读取到换行符(\n)为止,可以理解为读取到通俗意义上的回车键为止.如果返回的是空字符串,就代表读取到达了末尾.[size]参数用于设置读取的字节数.

3. readlines([sizeint])函数

读取所有的行,可以设置读取的字节数.

请注意:读取之后,光标的位置是会移动的.

2. 写入函数write序列

1. write([string])函数

将string写入文件,返回值为写入的字符数

2. writelines([string])函数

写入多行数据,自行换行.

3.4.2.4 光标位置相关函数

为了在适当的位置写入数据,我们需要合理控制光标的位置.

1. tell()函数

获取当前光标的位置,是一个相对于文件开头位置的偏移值,是一个整数.

2. seek(offset[, whence])函数

o offset参数

需要的偏移量,如果是负数,就从文件末尾开始.

o whence参数

更改偏移量的起点位置.

0从文件开头开始偏移

- `fileName.seek(0, 0)` 是开头的字母原因是在开头的位置偏移0位,所以是第一个字母

1从现在光标所在位置开始偏移

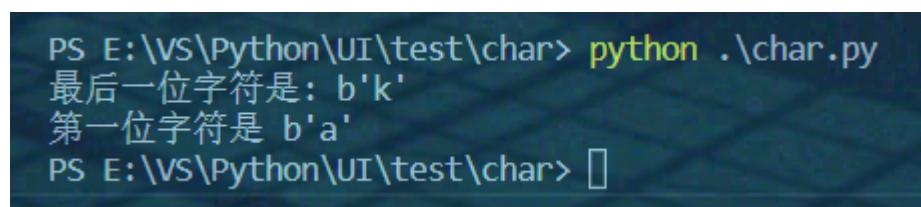
- `fileName.seek(2, 1)` 在上一句的前提下,是3个字母.

2从文件末尾开始偏移

- `fileName.seek(-1, 2)` 无论是否在以上两句的前提下,都是最后一个字母

```
1 # 获取文件的倒数第5个字符
2 getFile=open('out','rb+') # 此处使用了二进制打开
3 getFile.write(b'abcdefghijkl') # b表示使用二进制写入数据.
4 getFile.seek(-1,2)
5
6 print("最后一位字符是:",getFile.read(1))
7
8 getFile.seek(0,0)
9
10 print("第一位字符是",getFile.read(1))
```

输出结果如下:



```
PS E:\VS\Python\UI\test\char> python .\char.py
最后一位字符是: b'k'
第一位字符是 b'a'
PS E:\VS\Python\UI\test\char> []
```

3.4.2.5 字符串处理部分

字符串处理部分我暂且不提,因为这个部分非常复杂,细节很多,需要花费大量的篇幅来说.

3.4.3 常用的三个模块

学到这里,相比你已经和模块打了一部分的交道了,无论是 3.1.2.2 中的 `tkinter` 还是 `pyinstaller` 模块,都是我们非常常用的.

我将介绍初学者最常用的三个模块,它们分别是 `time` 模块(有时候使用更高级的 `datetime` 模块), `random` 模块, `math` 模块.

3.4.3.1 time模块

时间模块其实没有太多要讲的,请自行了解吧

3.4.3.2 random模块

导入random模块: `import random`.

3.4.3.2.1 常用函数

1. seed()函数

用于设置随机数种子.随机数的产生并不是真正的随机,而是伪随机,所以我们需要设置一个种子,通过算法,生成对应的随机数.

```
1 | random.seed(1111)
```

设置种子为1111.

2. random()函数

生成一个[0.0, 1.0)范围内的随机浮点数(小数).

3. randrange()函数

生成指定范围内的一个随机整数,语法结构和 `range()` 函数完全一致,不讲了.

4. randint(a, b)函数

返回随机整数在[a,b]之间.

5. uniform(a,b)函数

返回随机浮点数在a和b之间,a,b大小无顺序.

6. normalvariate(a,b)函数

正态分布函数,a平均值,b标准差

3.4.3.2.2 特定范围的随机数生成

在仅使用random()函数的前提下,其实已经可以生成任意范围内的随机数了.

比如生成100到150之间的随机数.

```
1 | rand=int(random.random()*50+100)
```

如果使用这种方法,关于取值区间左右开闭的问题还请自行了解.但是python既然提供了相应的函数,就没有必要自己使用这种方法了,只需要知道即可.

3.4.3.3 math模块

最常用的数学常量,常量被定义在math中, `math.pi=3.141592653589793`.

常用函数列表:

函数名	作用
<code>math.factorial(x)</code>	返回x的阶乘
<code>math.fabs(x)</code>	返回x的绝对值
<code>math.sqrt(x)</code>	返回x的平方根
<code>math.pow(x, y)</code>	返回x的y次幂
<code>math.exp(x)</code>	返回e的x次幂
<code>math.cos(x)</code>	返回x弧度的余弦值
<code>math.sin(x)</code>	返回x弧度的正弦值
<code>math.tan(x)</code>	返回x弧度的正切值
<code>math.degrees(x)</code>	将角度x从弧度转换为度数
<code>math.radians(x)</code>	将角度x从度数转换为弧度
<code>math.floor(x)</code>	将x向下舍入到最接近的整数
<code>math.ceil(x)</code>	将x向上舍入到最接近的整数
<code>math.trunc(x)</code>	删除小数部分,获取整数部分
<code>math.isqrt(x)</code>	将平方根数向下舍入到最接近的整数
<code>math.fmod(x, y)</code>	返回x/y的余数
<code>math.log(x[, base])</code>	使用一个参数,返回 x 的自然对数(底为e)
<code>math.log10(x)</code>	返回x底为10的对数

实际上以上部分函数并不是必须的,但是我依然写了出来.

`sin()`,`cos()`,`tan()`我最常用的只有`sin()`.

`sqrt()`,`pow()`,`exp()`我都使用了`pow()`.

`log()`,`log10()`相比,`log10(x)`比`log(x, 10)`更加精细,类似的函数还有很多,但是我只选取了常用对数`lg`和`ln`.

3.4.3.3 我比较喜欢的模块

1. `tkinter`模块,用于绘制简单的gui,前两天看到了个新的GUI模块,说是包含`tkinter`,比他更强,但是还在测试,所以使用这个就很好,当然了最主要的原因是好上手.在 3.1.2.2 中使用的就是`tkinter`模块.
2. `manim`模块,用于绘制数学公式动画,油管上大量数学视频都是使用它来做的,质量不必多说.
3. `Matplotlib`模块,用于可视化python,开玩笑可视化超帅的杂乱无章的数据变成图表不帅吗.
4. `pymunk`模块,物理引擎,开玩笑谁不喜欢简单的2D物理引擎呢.
5. `xlwt`模块,处理excel表格用的,可以快速处理一些数据,或者根据表格生成代码用.
6. `json`模块,如其名,处理json文本用的,和`xlwt`用途类似.

7. re模块,正则表达式,无需多言.
8. urllib模块,访问网页用,可以下载文件
9. Bs4模块从html或者xml中提取数据的库,加上上面几个懂吧,网络爬虫走起.
10. NumPy模块,提供科学计算的包,只知道他名声在外,没用过,估计是搞科研用的.

截止到现在我推荐了几个比较常用的模块.尤其是前几个非常好玩.如果有你感兴趣的,就自己去学习吧!

3.5 实战 解析上文中的代码

3.5.1 输出0到n

题目:提供一个整数n,输出从0开始到n结束全部数字,包括n.

```

1 # 处理部分
2 def printNum(n):
3     for i in range(n + 1):
4         print(i)
5 # 输入部分
6 n = int(input("请输入一个数字 n:"))
7 # 输出部分
8 printNum(n)

```

这是我们以前提供的代码,我提到了他有问题,现在来修改.

题目中提到给的是整数n,自然没有说是正数还是负数还是0,我们不考虑非实数部分和小数部分.别和我说什么小数不是数字吗.

所以在 `printNum()` 函数中,我们应该分三种情况讨论.下面是修改后的代码和注释.

```

1 ## 处理部分
2 # 定义一个函数printNum,接受输入参数n
3 def printNum(n):
4     # 如果n大于0
5     if(n>0):
6         # 正数
7         # 使用for循环遍历数组
8         # 使用range()函数生成[0,1,...,n]的序列
9         # 开始值默认为0,步长默认为1,结束值n+1不含在内,所以为了输出n,应该使用n+1
10        for i in range(n + 1):
11            print(i)
12        # 如果n小于0
13        elif(n<0):
14            # 负数
15            # 题目要求从0开始输出,所以不能使用range(n,1)来简单的生成,固然可以生成我们需要的序列,但是顺序反了.
16            # 步长设置为-1,使数列呈递减,n-1还是因为n-1不可取.
17            for i in range(0,n - 1,-1):
18                print(i)
19        else:
20            # 0 非正非负即为0,没什么好说的
21            print(n)
22 ## 输入部分
23 # 使用一个主循环可以让程序一直执行
24 while(True):
25     # 使用input函数来接收输入的数字,并使用int()函数将其转化成为int类型

```

```
26     # 其实没必要,因为输入的必然是个整数.  
27     n = int(input("请输入一个数字 n:"))  
28     # 输出部分  
29     printNum(n)
```

3.5.2 记录点击次数的按钮

以前我是直接提供了代码,现在我试图解释这个代码是怎么写出来的.我将以tkinter为例子,简单的介绍如何使用陌生的模块.

[tkinter的中文手册](#)

[tkinter的官方教程](#)

3.5.2.1 分析和拆解需求

一般来说,我们需要分析目的是什么,然后为了实现目的我们需要什么工具,这些工具我们可以配置些什么?然后我们怎样使用这个工具来达成目的呢?

思路总的来说如上.我们来具体到tkinter中去.

目的是记录点击按钮的次数,为了实现这个目的,我们必须有的是一个窗口,上面有一个按钮和一个点击次数的文本.所以我们需要三个工具.

1. 窗口用来承载按钮和文本
2. 按钮用来进行点击行为本身
3. 文本用来显示我们点击的次数

所以我们需要创建一个窗口,然后查询这个窗口我们可以设置些什么?随便点开一个窗口,比如edge浏览器,我们观察到主页标签有**新建标签页**字样,然后窗口本身没有全屏显示,所以窗口必然还有个大小.

也就是对于窗口,我们需要设置**大小**和**标题**这两个参数.

然后我们在窗口上创建按钮.

按钮有什么呢?我们回想任意一个窗口上点击确认的按钮,按钮属于哪个窗口?它上面显示了确认还是取消?文字的颜色是什么?背景色是什么?点击它后执行了什么事情?

则有对于按钮我们有**父窗口**,**显示文字**,**文字颜色**,**行为**等等这几个参数需要设置.

再来看显示的文本,显示的文本自然就是普通的文本,他和word文档中的参数并没有什么不同,所以参数自然是一样的.而按钮的文本和显示文本没有区别,所以我们可以把**文字颜色**和**显示颜色**两个参数合并为**文本参数**.

那么我们有了如下结构

- 父窗口
 - 标题
 - 大小
- 按钮
 - 父窗口
 - 文本参数
 - 行为
- 显示文本
 - 文本参数
 - 背景色

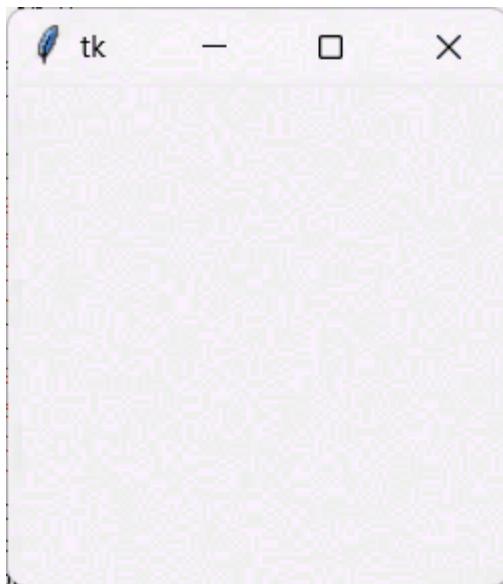
3.5.2.2 写核心代码

依照上面的结构,我们开始写代码.

首先输入模板,使用一个库之前必须import它,我这里还起了个别名叫tk.

```
1 import tkinter as tk  
2  
3 window=tk.Tk()  
4  
5 window.mainloop()
```

执行后可以看到空窗口



第一句话创建窗口,第二句话开始主循环显示窗口.

现在开始设置窗口的属性,一个是标题,一个是大小

```
1 window.title('ClickLog')  
2 window.geometry('300x300')
```

添加文本在窗口上吧,所以添加一个文本组件 Label.

```
1 text = tk.Label(window, text="这里会显示你点击了几次按钮!", background="yellow",  
2 fg="black", font=('Times', 15, 'bold'))  
text.pack()
```

Label 控件的参数如下 父窗口,参数列表 .

父窗口设置window,就是我们之前创建的那个窗口,随后设置参数,这里我们设置显示的文本(text),背景色(background),文字颜色(fg),字体(font)这几个参数.

随后使用 text.pack() 命令让他显示在窗口上.

对于 button 这个控件而言,参数列表和 Label 类似.

```
1 button = tk.Button(window, text="点击我!", command=callback)  
2 # 将按钮组件添加到窗口上  
3 button.pack()
```

我们需要注意的是 command=callback 这个参数,我们调用了 callback 这个函数.所以现在我们来写这个函数.

函数的作用是显示点击文本的次数,所以我们很容易想到这个函数的行为

1. 点击的时候点击次数+1
2. 更改显示文本

所以我们就有了两句话 num += 1 表示这个计数器加一. text.configure(text=f"这是你第{num}次点击按钮") 来更改文本显示的内容.

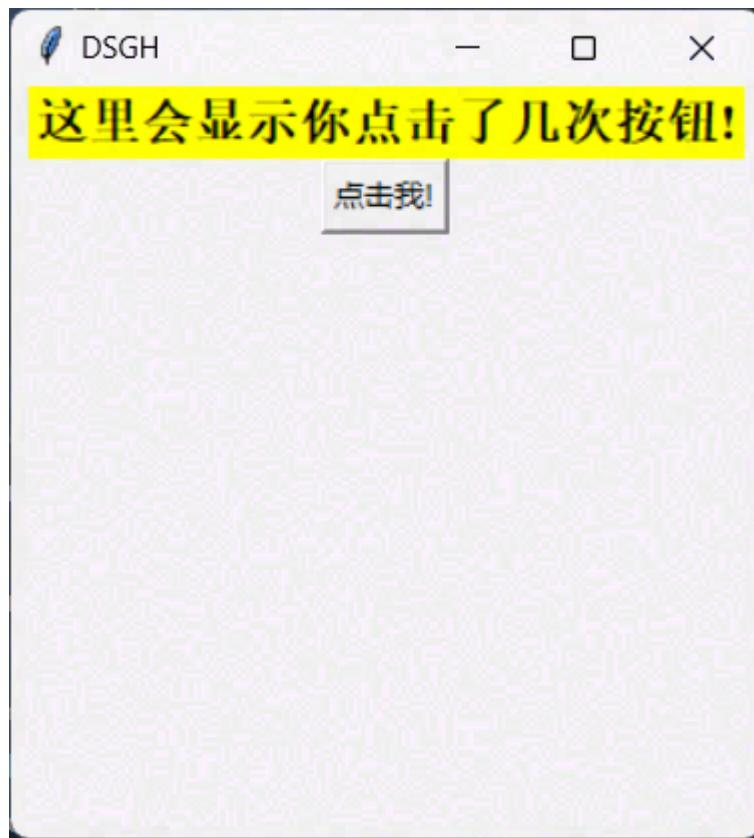
随后我们考虑到num这个变量没有定义,所以在主函数(callback函数外)将他设置为0,并在callback函数内部将他设置为全局变量方便修改.

```
1 num =0
2 def callback():
3     global num
4     num += 1
5     text.configure(text=f"这是你第{num}次点击按钮")
```

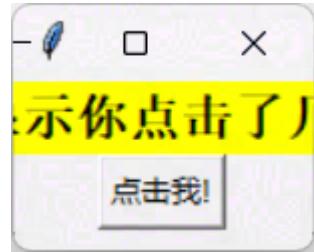
至此这个程序就应该结束了,接下来只需要进行程序的主循环就可以保证程序正常运行了.

3.5.2.3 还需要改进的地方

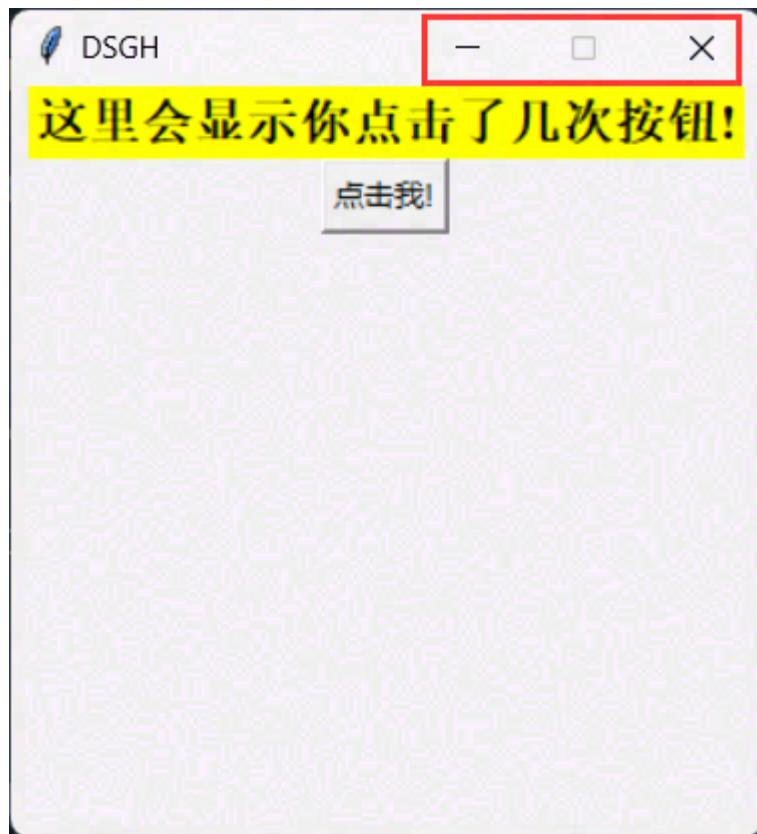
我们虽然已经实现了完整的功能,但是这个程序依然不成熟,例如我们没有设置各个控件的为止,我们使用的都是默认设置,也就是窗口如下显示.



他的问题在于如果用户手动改变了窗口的大小,会有不符合我们预期的情况出现.比如下面这样.



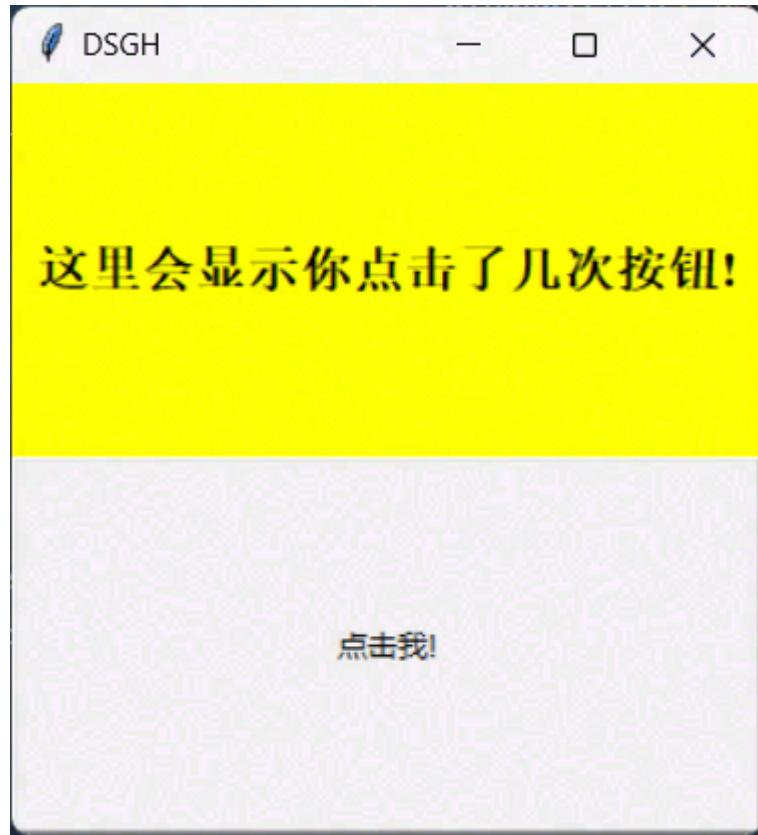
label 组件没有被正确的显示在窗口中.我们当然可以简单的禁止用户改变窗口大小,如下,窗口甚至不能最大化,自然控件的为止就不能错位了.



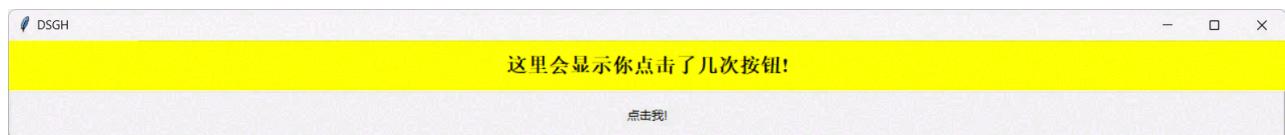
如上,相比于第一个窗口,以上的程序中最大化按钮是灰色的禁止使用状态.

不过这里我更加建议各位通过改变控件的布局来避免这个问题,这样也可以学习到如何控制各个控件的位置相关的知识.

比如下面这样.



简单的设置了他的布局和属性.这样即使他被用户更改的奇形怪状,也可以正常显示.



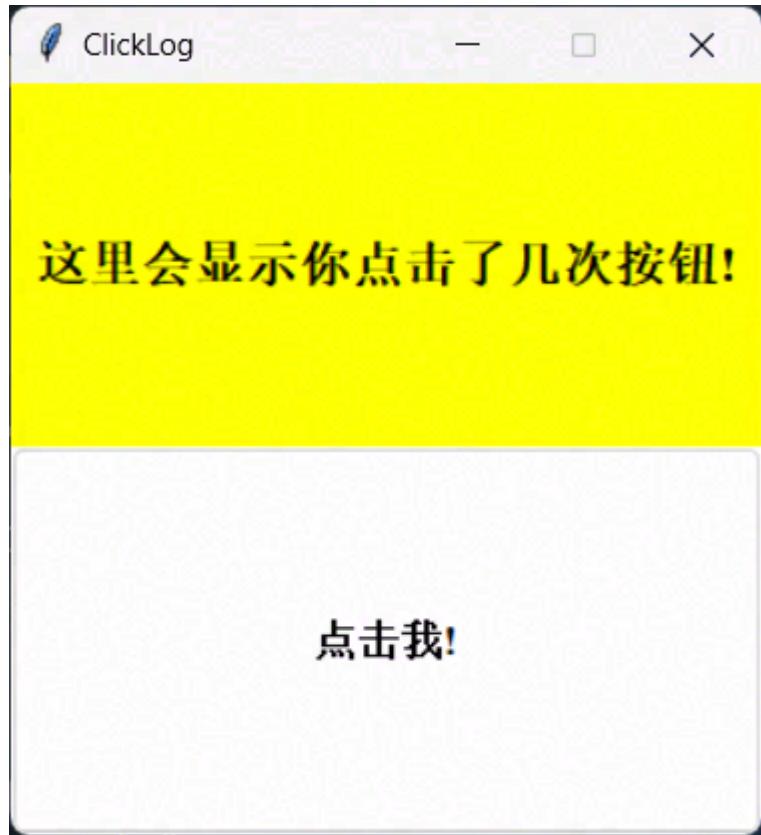
同样的,字体的大小也可以进行设置.这里就不展开来说了.

还是禁止用户改变窗口大小方便.

我们看到这个窗口的按钮边框有阴影,这是tkinter这个组件的缺点,所以我们可以使用ttk来改进这个程序.改进的程序还请自行学习ttk技术来实现.

实际上tk已经不再常用,我所展示的这个程序是我在四年前在某个不知名小网站上学习tkinter时写出来的古老东西,当时我还不知道ttk,后来我知道了ttk后就再也没有碰过这个程序了()

使用ttk重写后如下,效果如下.



3.5.2.4 原代码提供

如下提供修改前的代码的所有注释.

```
1 # python无法导入`pyi_splash`模块,所以使用`try-except`语句跳过这个异常.
2 try:
3     # 导入模块
4     import pyi_splash
5     # 关闭闪屏画面
6     pyi_splash.close()
7 except ImportError:
8     # 跳过
9     pass
10
11 # 引用tkinter模块,并取别名为tk
12 import tkinter as tk
13
14 # 创建一个窗口
15 window = tk.Tk()
16 # 修改窗口名称为ClickLog
17 # 写到这里突然发现显示的标题是DSGH,之前截的图我真的懒得改了,所以就这样吧.
18 window.title('clickLog')
19 # 设置窗口大小
20 window.geometry('300x300')
21
22 # 定义记录的初始值为0
23 num = 0
24
25 # 定义函数,以便于在点击按钮的时候调用
26 def callback():
27     # 全局变量num,便于我们修改num的值
28     global num
29     # 自增,num += 1等效于num = num + 1
```

```

30     num += 1
31     # 更改显示文本的内容为"这是你第几次点击按钮"
32     # f表示格式化文本,将{num}表示为占位符,在现实数据的时候使用num的值来替换{num}达到显示次数的目的。
33     text.configure(text=f"这是你第{num}次点击按钮")
34
35     # 添加一个文本组件设置父窗口为window
36     # 更改显示文本为"这里会显示你点击了几次按钮!"
37     # 设置背景色为黄色
38     # 文字颜色为黑色
39     # 设置字体,并修改大小为15,加粗
40     text = tk.Label(window, text="这里会显示你点击了几次按钮!", background="yellow",
41                      fg="black", font=('Times', 15, 'bold'))
42     # 将文本组件添加到窗口上
43     text.pack()
44
45     # 添加一个按钮组件,设置父窗口为windo
46     # 修改显示文本为"点击我!"
47     # 设置点击时调用callback函数(即上文定义的函数)
48     button = tk.Button(window, text="点击我!", command=callback)
49     # 将按钮组件添加到窗口上
50     button.pack()
51
52     # 进入主循环
53     window.mainloop()

```

3.6 星辰大海 未来你可能会遇到什么

3.6.1 语法部分

关于语法部分,实际上我仅仅介绍了最基础的一部分,我们还有很多很多可以去学习,最简单的是**列表,元组,字典,组合**这四个数据类型,他们会频繁的出现在我们的日常使用中,没有他们,很多复杂的算法和程序就无法实现.

然后你还需要学习**推导式**,他们可以帮助你很容易的生成以上的四种类型变量.随后你还会学到**迭代器和生成器**.

你会接触到**函数**,并体会函数如何化简了你的程序,是你的程序更加直观,更加易于使用.随后你还会遇到**lambda表达式**,去思考如何使用它来化简你的函数.到了函数部分,你还会接触到**变量的作用域**问题,这样你就会明白为什么我在 3.2.2.4.1 中有一句 `global num`.

终于,语法部分你学习的差不多了,剩下的内容已经不在挡手.

3.6.2 面向对象技术

接下来你会学习面向对象技术,你会听到那一句真理**万物皆对象**,去体会**面向对象编程**思想和**面向过程编程**思想的不同.

你会创建一个**类**,然后将它**实例化**,你会调用这个**对象中的方法**来实现种种功能等等等等.给我讲激动了(bushi).

3.6.3 什么是编程?

编程=算法+数据结构.

你会学习到种种算法,你会逐步开始学习各种各样的算法,排序,深搜,广搜等等.然后你会去学习堆栈树图等等.

此处我并不会多说,因为我不认为我现在有资格去讨论这部分的内容,所以....加油~

3.6.4 如何学习编程?

我永远认为学习编程的最快的方法就是**阅读文档+实践操作**.

知识点可以迅速的过一遍,尤其是像python这种简单的语言,他的基础语法非常简单,我们可以花上几个小时把基础语法过完,然后迅速开始实践,编程不同于其他的科目,他的实践没有什么高成本,他不是我们一年去一次的实验室,而是与我们365天相伴的电脑上的一角,所以我们可以轻松的在任何时候去进行实践.

我在上文中也说过,不要轻信他人的结论,我们在物理化学上因为无能为力去进行实验验证某些结论,但是对于计算机编程而言,大部分的东西我们都可以亲自实践(不要给我整那些算力的硬件问题啊喂).

在实践过程中我们会遇到很多问题,我们会通过搜索引擎去解决这个问题,或者是偷个懒,和ai碰个头,去解决这个问题,但是无论如何,这个过程中我们的解决问题的能力都在上升,然后我们可以再反过来去阅读文档,会发现更加理解文档.理解文档后会实践更轻松,这就是一个正向循环.

随着这个过程的进行,你的解决问题能力会上升,动手操作能力会上升,阅读文档的能力也会上升,逐渐的,你就可以自己去写一些文档.我想这也是乐趣所在.

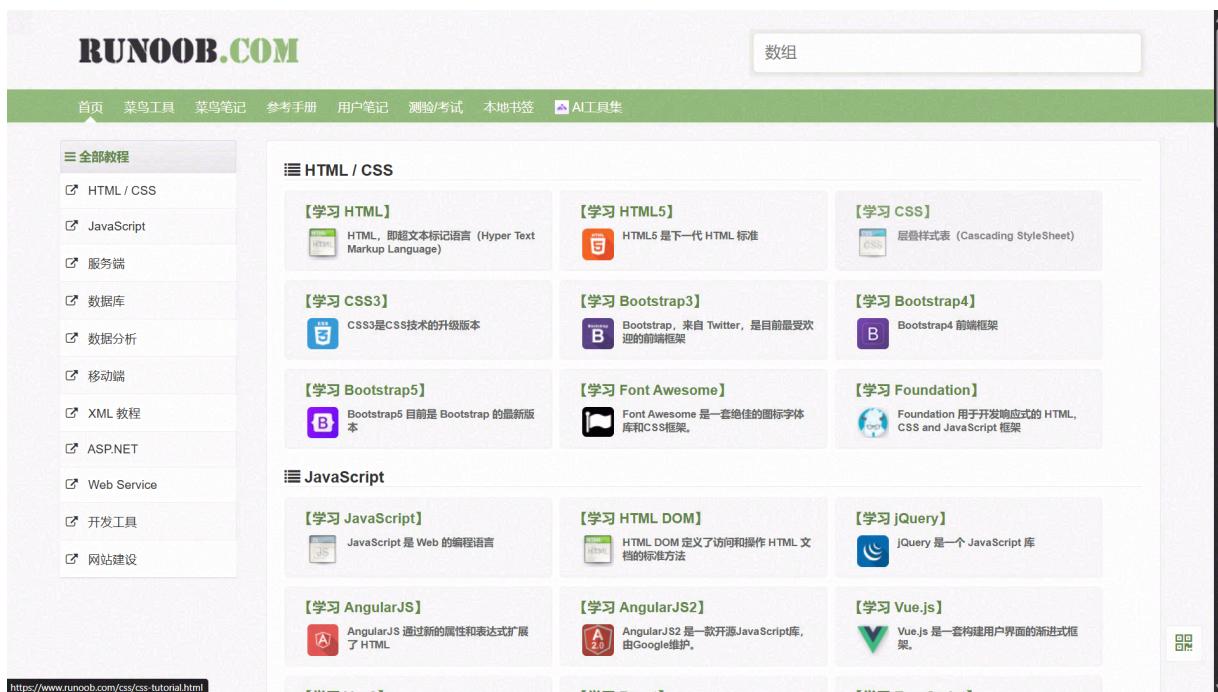
总之学习编程的最好方法在我看来是**理论+实践双循环**.

3.7 自学python我想给你点什么

3.7.1 好的教学网站

1. 菜鸟教程

学的不仅仅是技术,更是梦想!



The screenshot shows the homepage of RUNOOB.COM. The top navigation bar includes links for 首页, 菜鸟工具, 菜鸟笔记, 参考手册, 用户笔记, 测验/考试, 本地书签, and AI工具集. A search bar at the top right contains the text "数组". On the left, there's a sidebar titled "全部教程" with categories like HTML / CSS, JavaScript, 服务端, 数据库, 数据分析, 移动端, XML 教程, ASP.NET, Web Service, 开发工具, and 网站建设. The main content area is divided into sections for "HTML / CSS" and "JavaScript". Under "HTML / CSS", there are cards for "学习 HTML", "学习 CSS", "学习 Bootstrap3", "学习 Bootstrap5", "学习 Font Awesome", and "学习 Foundation". Under "JavaScript", there are cards for "学习 JavaScript", "学习 HTML DOM", "学习 jQuery", "学习 AngularJS", "学习 AngularJS2", and "学习 Vue.js". Each card includes a small icon and a brief description.

○ 优点

1. 在线免注册,无广告收费,干净到极点的网站.
2. 教程简洁干练只讲最需要说的东西,正做到了想学什么就学什么.(这是优点也是缺点)
3. 分类合理,不会出现浪费时间学习到与目的不符的知识的情况.不会出现像我一样学了半个月java才发现问题,转头开始学习c#的情况.
4. 更新及时,技术都是最新的.也没有什么杂七杂八的社区,影响学习氛围.

○ 缺点

1. 教程简洁干练只讲最需要说的东西,正做到了想学什么就学什么.(也就是做到了必要的省略)
2. 纯粹的理论说实话看久了会感觉干的被噎到.

3. 真的就是关于编程知识的网站,如果你想要学习电脑或者软件的知识,那你只能扫兴而归.(除了linux和git).
4. 这里只有理论,只提供学习资料,基本不帮助你解决问题.

安卓端APP[下载网址点击此处](#).

安卓端APP二维码如下



菜鸟工具界面我并没有详细使用过我不做评价,是否

2. [W3CSchool](#)

编程狮

The screenshot shows the homepage of W3CSchool. At the top, there's a navigation bar with links for '编程课程' (Programming Courses), '编程实战' (Practical Programming), '学习路径' (Learning Paths), 'VIP会员' (VIP Member), '编程题库' (Coding Question Bank), '编程教程' (Programming Tutorials), '在线工具' (Online Tools), and '严选训练营' (Strategically Selected Training Camps). A search bar is located at the top right. Below the navigation, there's a large banner for '带领女程序员小白:一小时轻松入门GIT' (Lead female programmers from zero to GIT in one hour) with a '免费试学' (Free Trial) button. To the right of the banner is a login form for '快速登录' (Quick Login) with fields for '学号/手机/邮箱' (Student ID/Phone/Email) and '登录密码' (Login Password), and buttons for '立即登录' (Log In), '微信登录' (WeChat Login), 'QQ 登录' (QQ Login), 'Github登录' (Github Login), and '注册账号' (Register Account). On the left, there's a sidebar with '新上好课' (New Good Courses) and categories like '前端开发', 'Python开发', 'Java开发', '后端开发', '编程基础与数据库', and '其它方向'. At the bottom, there are sections for '实战训练' (Practical Training) and '学习路径' (Learning Paths) featuring courses for Python, Frontend, Java, and C/C++.

○ 优点

1. 内容比菜鸟丰富很多,不仅仅局限于编程语言的教学,也关注到了开发过程中的各方各面.
2. 开发手册同样是免费且干净的可以保证开发手册内部没有广告.但是网站偶尔会有广告弹出来.
3. 其余部分和菜鸟类似.

○ 缺点

1. 毕竟是以盈利为目的的商业网站,和菜鸟的定位就不同,也只有开发文档是免费的.不过其他的网课和所谓的训练营不管的话,那就无所谓.

2. 教程地狱现象明显(不够简洁)

以python教程为例下面是他的目录.



可以明显看到有没有必要的东西进入其中,和B站上盲目堆积时间的200小时或者1000集的教程没有区别.

小白很容易迷失在茫茫学海里,例如数据分析和web开发放在了一起,这两个并不是必须都学不可.

所以更加推荐把W3Cschool手册当成字典去查阅,而不是学习,毕竟没有人对着新华字典学习中文.

3. 获取部分资源名义上免费,实际上需要添加微信公众号获得验证码等等

4. 离开主站之后的分站手册页面广告就不少了

3. [Python官方文档](#)

官方文档无需多言,最即时也是最权威的网站.

- 优点

1. 权威且及时,可以看到任意版本的详细信息.
2. 严谨且详细,开发文档会把每个参数都告诉你,无论是否常用
3. 有详细的[新手入门指南](#),官方一步一步喂饭吃.

- 缺点

1. 需要较高的阅读文档水平才可以保证有收获,小白可能没有耐心或者看不懂文档.
2. 部分内容翻译不及时或者不准确,容易产生歧义

- 克服以上缺点

1. 锻炼阅读文档能力,可以先从简单的项目文档开始入手
2. 借助ai,ai可以极大程度的简化我们的编程学习过程,利用ai进行文档阅读事半功倍.

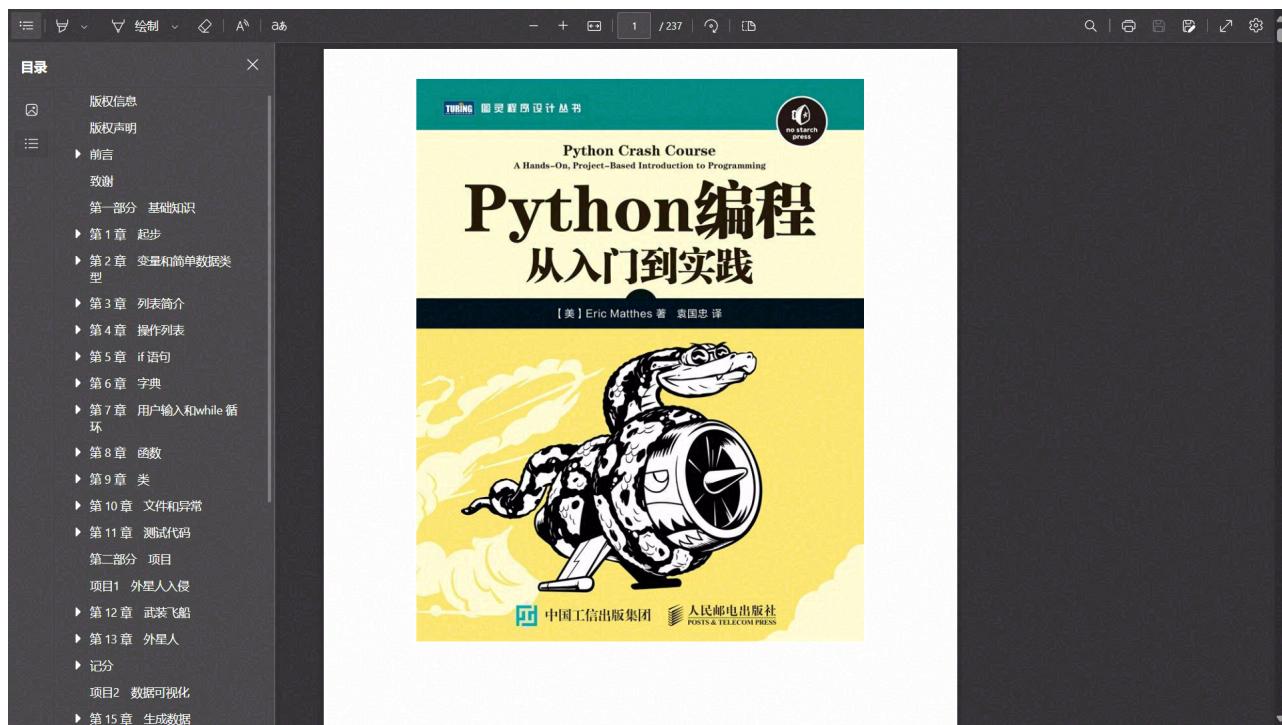
4. [Python算法学习](#)

github上的一个项目,使用python完成了全部的算法.

The screenshot shows a GitHub repository page for 'The Algorithms - Python'. The repository has 15.8 MB of code and 282 online contributors. It features a logo with two interlocking 'A's and the text 'The Algorithms - Python' and '算法 - Python'. Below the logo, it says 'All algorithms implemented in Python - for education' and '所有用 Python 实现的算法 - 用于教育'. A note states that implementations are for learning purposes only and may be less efficient than standard library implementations. The 'Getting Started' section includes a link to 'Contribution Guidelines'. The 'Community Channels' section lists several links. The repository was last updated 5 months ago.

3.7.2 书籍推荐

因为python我是简单的学一下,所以书籍我只读了一本,所以我也只推荐这本书.



python编程从入门到实践.

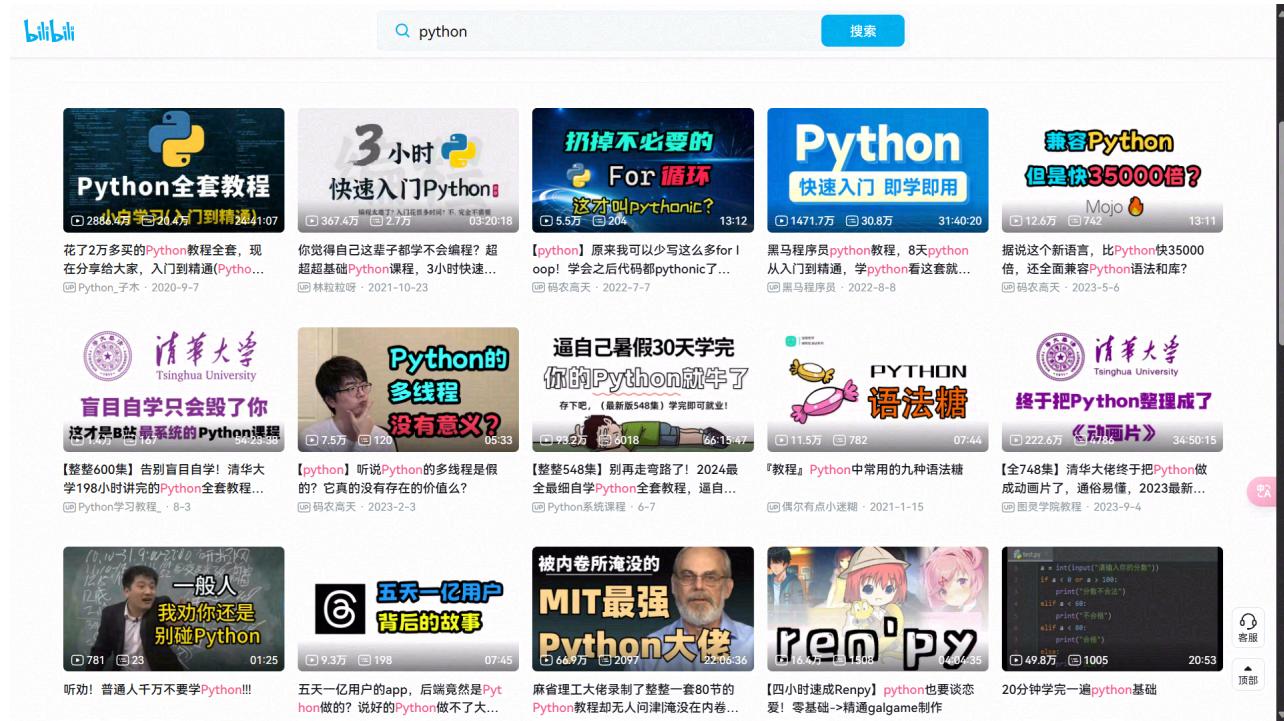
我对这本书的评价挺不错的,可以说是一本好书,最重要的是,他用很少的篇幅就讲完了这么多,这本书居然才250页!

3.7.3 视频教程

我实在是不建议新手使用视频来学习python,原因很简单,以b站举例,上面盛行各种**花多少W买的课程,900集,一个星期从入门到精通,清华大佬多久讲完,冒着被开除的风险,已经被开除等等**这样吸引人的话术,更别提什么**学会爬虫,月入过10W**这样的视频了.那这样的视频,他们动辄就五六个小时来上课,可实际上这些课真的值得你去花费五六个小时去看吗?我曾经看过一套这类型的教程,说实话,无非两种,第一种是国内的培训机构的录制课程视频,这种食品质量参差不齐,有的好有的坏,还有一种就是从外网搬运过来的各种视频,然后缝合在一起,剩下的就是站内重复搬运,从这个up上下载下来,再从这个up上换个名字上传上去.

很多小白在看这种课程的时候没有筛选性,这种200集的课程往往是由多个部分组合而成,基础篇,进阶篇,爬虫篇,自动化篇等等,可是实际上大部分人仅仅只需要学习一个基础篇的内容就已经足够了。

这就是**教程地狱**,什么都教,等于什么都不教。我曾经看过一个话题他说B站真的称得上是大学吗?热评第一让我觉得很有道理,他说如果把油管叫做大学,那么b站最多是个中专。我并没有说崇洋媚外,推崇油管多好的意思,但是我认为b现状正是这样.大量杂乱无章质量参差不齐的教程堆积在b站中,随后劣币驱逐良币,站内的一些汉化组所翻译得到授权的精品课程就没有什么人去看了,随后人们再理所当然的不愿意为知识付费,这样的b站不是中专是什么呢?



而如果你在油管上去搜索,那么开头的两个视频的市场不过是4个小时和6个小时.



往后再长的视频也不过是15个小时.



Python Full Course for free

18M views • 3 years ago

Bro Code

Python tutorial for beginners full course #python #tutorial #beginners ★Time Stamps★ #1 (00:00:00) Pyt...

100 chapters 1.Python tutorial for beginners | 2.variables | 4.string methods ↻ | 5.typ... ▾



Harvard CS50's Introduction to Programming with Python – Full University Course

4.7M views • 1 year ago

freeCodeCamp.org

Learn Python programming from Harvard University. It dives more deeply into the design and implementation of web apps with ...

4K

而这套课程的内容也不过是花费了10到20分钟介绍了python可以做到每件事.

小白需要知道的是学习这些有什么用,而不是在学完了后才意识到这个东西学了没有.

以下这套课程是我推荐的书籍的随书代码,由图灵社区提供.图灵社区的地址我同样贴在下面,毕竟我确实在图灵上看了不少书.

[【官方随书视频】最热门编程入门书《Python编程：从入门到实践（第2版）》](#).

[图灵社区](#).

随后的这套课程是我比较推荐的,他只用了2个小时把python的基础讲完了,而且真的只讲了基础,但凡进阶的一点不说(笑).我当时是当作电影一口气看完的,这套课程也确实引起了我的兴趣,让我把上面的拿本书读了下去.

[《Python负基础到入门教程》专为"非计算机专业和编程困难户"制作（全13集 配音字幕重制版）](#).

他的其他视频也可以帮助你获取一些计算机基础知识,最重要的是,他真的是做到了**负基础**入门.

总之,如果可以翻墙的话,在油管上去学习python是最好的选择,如果不借,借助上面的第一套课程入门python,随后以我推荐的书为核心,配合练习视频就可以把python学个大差不差了.

3.7.4 善用AI

ai可以帮助我们学习python,菜鸟教程中提供了一个code的ai插件,可以选择他试一试.

chatgpt-4会是更好的选择,实测chatgpt-3.5编程效果不佳,需要花费不少的笔墨去完善它.

能够使用new bing或者copilot也可以.另外一个值得推荐的ai是克劳德.

国内的ai的话我用得不多,我只用过阿里的通义千问和百度的文心一言,所以我就不推荐了.

我们可以把不会的问题当作和人对话一样直接问ai,他会给出解决方案,如果解决方案不行,你就进一步和他说,他会和你修正.

如果程序出现了无法理解的报错,直接把整个报错复制给ai,他会给你解决方案.

如果程序没有报错,但是运行结果不符合预期,直接复制整个程序给ai,告诉他你的问题,他会帮你修正.

但是不要过度相信ai,ai有时候解决问题真的还没有搜索引擎快.

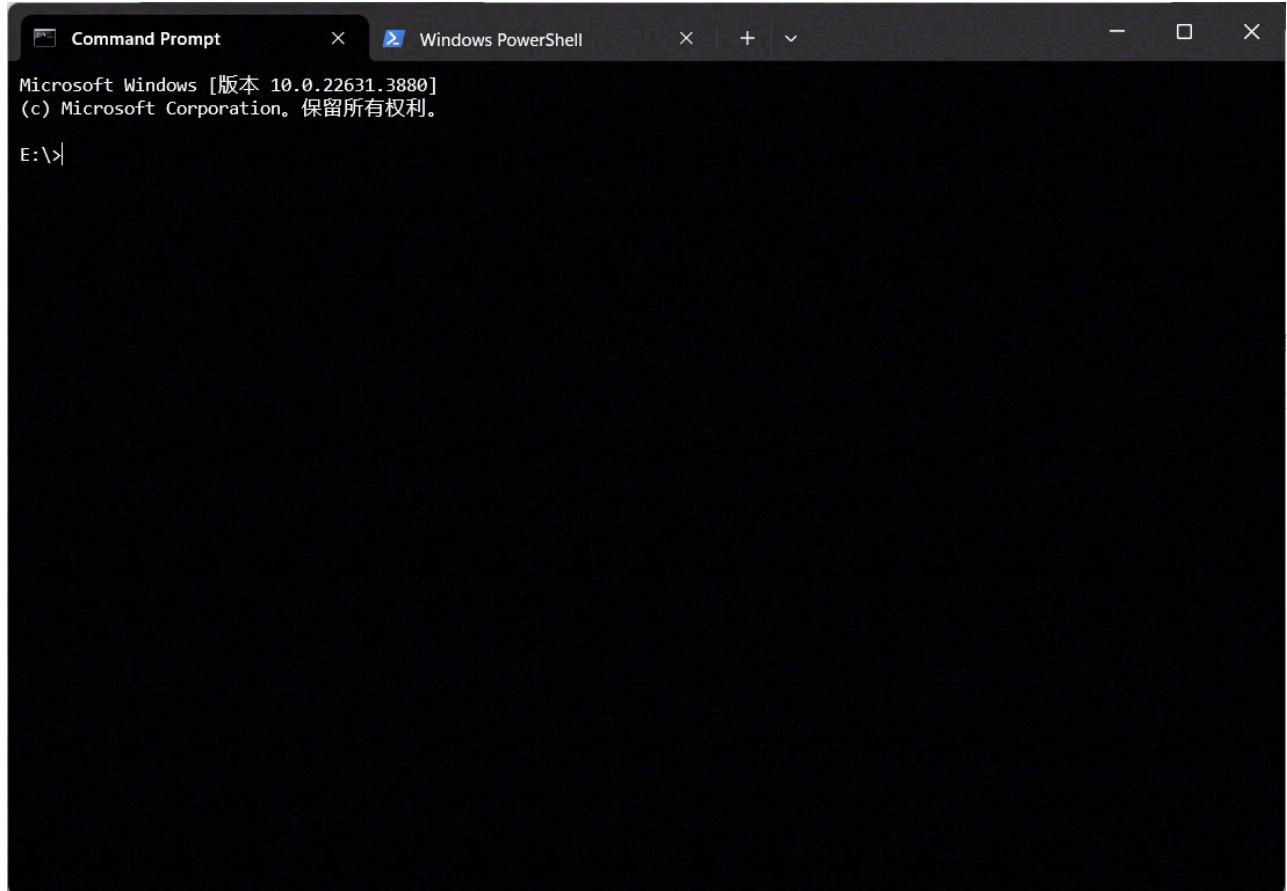
4 Windows终端

4.1 CMD和Powershell

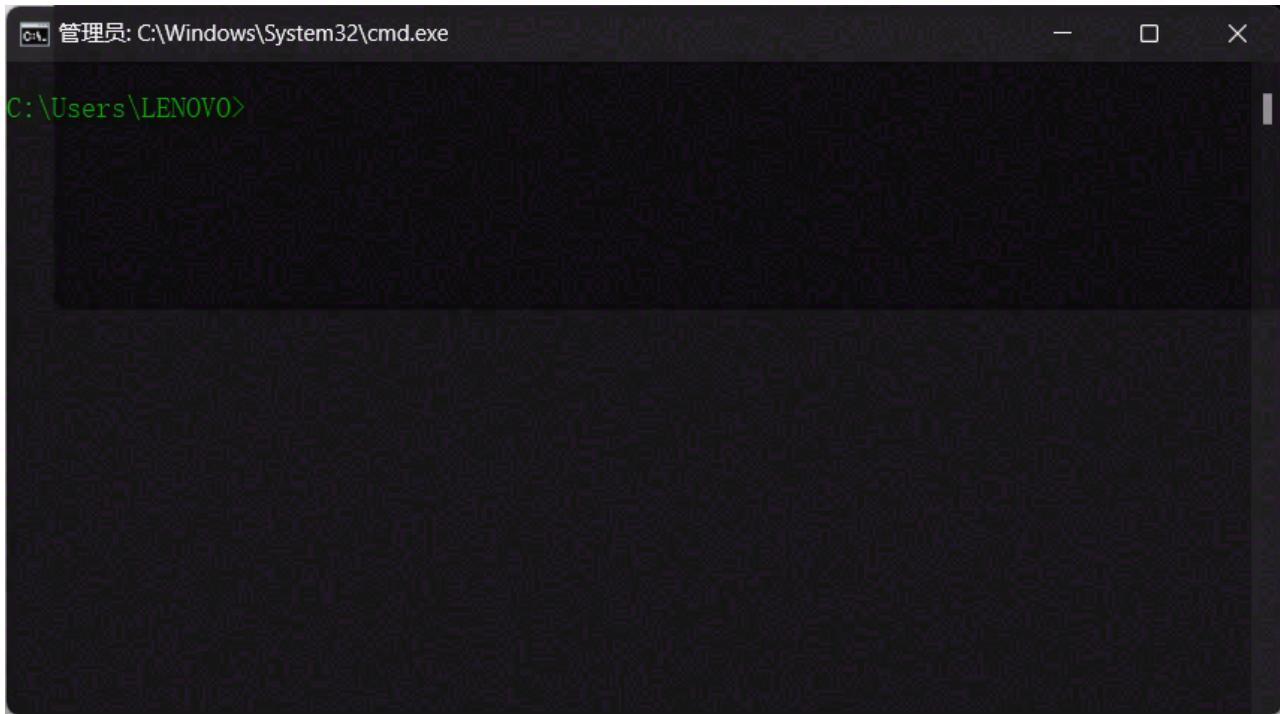
windows下有两个终端,第一个叫做CMD(command),另一个叫做PowerShell.我们来介绍他们的区别和使用的选择.

打开终端界面,win11新版本将cmd和powershell均合并到了终端中,所以可以简单的通过打开其中任意一个打开终端.实际上打开cmd还是powershell都可以().

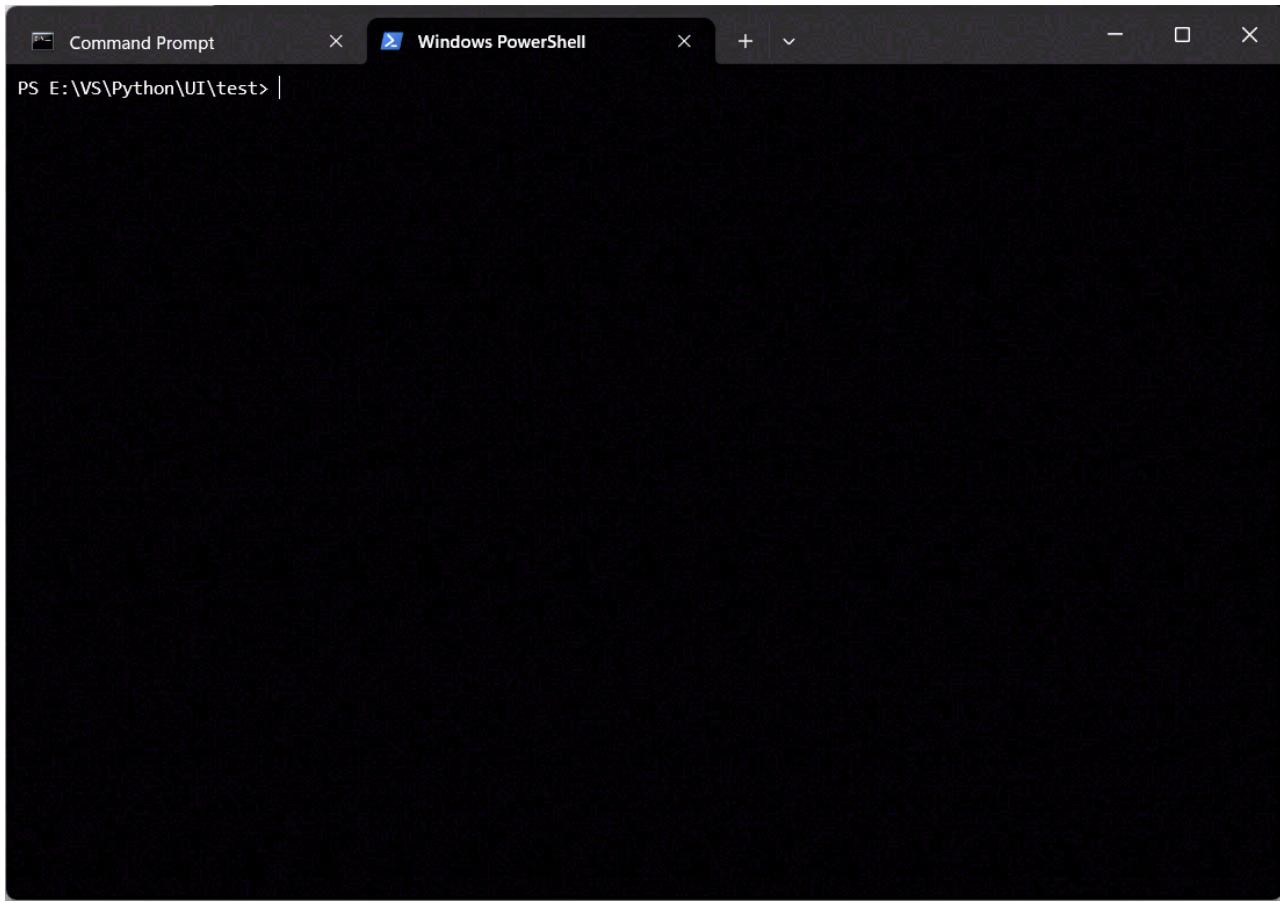
终端下的cmd窗口.



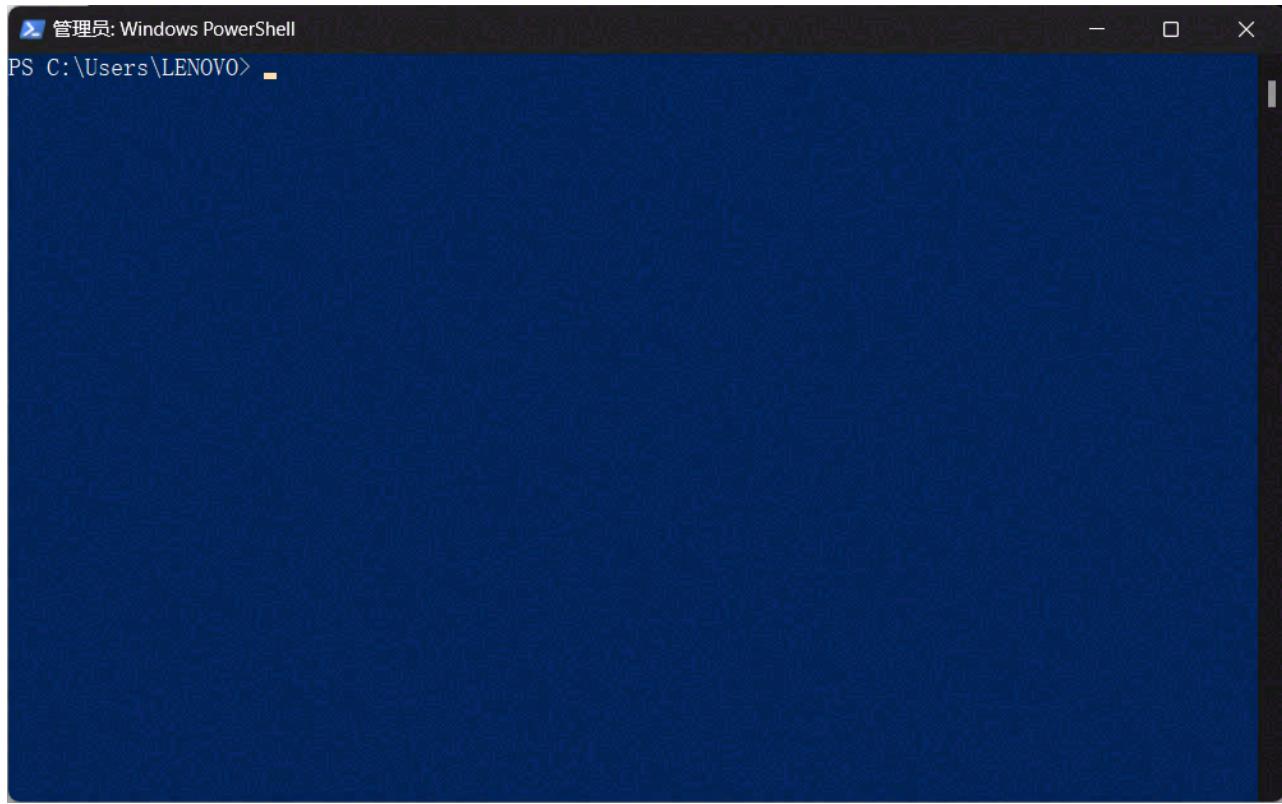
普通的cmd窗口



终端下的powershell窗口.



普通情况下的powershell



终端的优点在于多标签页.他和浏览器一样使用了多标签页的功能,当然他没有浏览器那么智能,但是也足够使用了.比如你可以看到我们上面的界面中终端有两个标签页,第一个是cmd,第二个是powershell.

如何区分现在自己使用的是cmd还是powershell?

我们可以观察命令行的提示符cmd的命令行提示符是由 `所在路径>` 构成的,而powershell则是由 `PS 所在路径>` 构成的.

也就是说如果有了PS字样,使用的就是powershell.

PS是powershell的简写,但是实际上很容易和Photoshop的简写PS弄混,所以更加常用的是psh作为简写.

一个回答就是cmd相较于psh更加冷冰冰的,而psh给人的感觉更具有感性.这并不是开玩笑,使用久了相比会感受到这一点吧.

cmd是一个交互窗口,我们在其中可以通过命令来和系统进行交互.而在资源管理器中可以进行的行为,这里也都可以,例如文件的复制粘贴,改名移动等等.但是这里也可以做到资源管理器中过不到的事情,所以cmd自然比资源管理器更加强大.

不过cmd是上古遗留的产物了.它的历史可以说是非常非常悠久,随着技术的发展,人们自然不会满足于使用一个冷冰冰的命令窗口,人们设计并制作了更加人性化的powershell.

powershell,可以简单理解为他是cmd的加强版,不过实际上底层完全不一样,包括命令也是使用了完全不一样的方法实现.cmd中使用的是简单的windows命令,但是psh中可以使用的则是 `cmdlet(command let)` 命令和 windows命令.所以一般来说psh可以作为cmd的上位替代来使用.

值得一提的是,虽然psh中也可以使用Windows命令,但是其实底层完全不一样,不如说微软讨了个巧,使用了一个种叫做别名 `alias` 的机制.后文我会提到他.

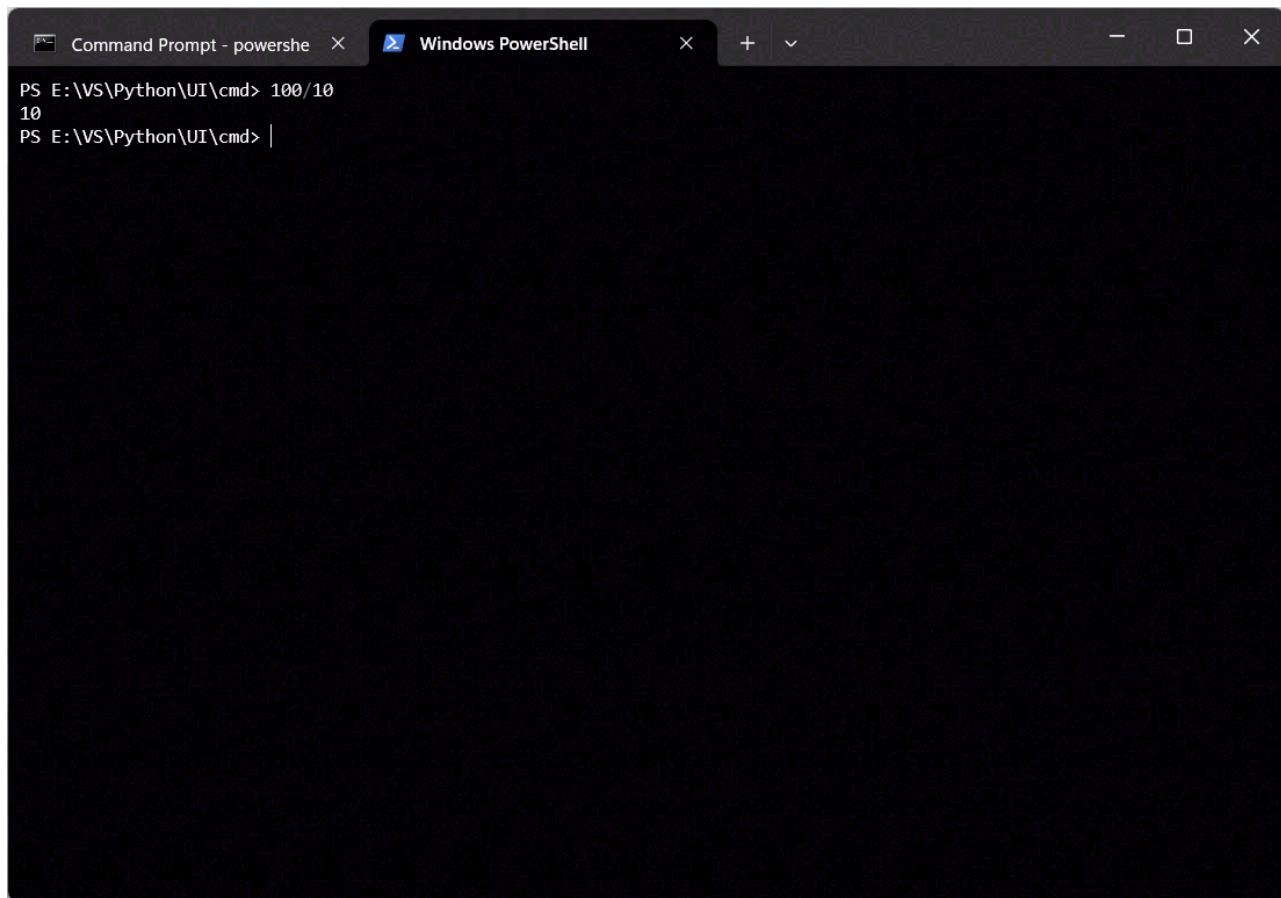
cmdlet的优点是好上手和易理解,cmdlet使用了 `.net` 库进行开发,使用 `.net` 库进行开发的优点是执行命令的返回值类型不同.

在pyth部分我们已经学习到了数据类型的概念,所以返回值理所当然也有数据类型的不同,cmd的命令返回值均为字符串,而psh的返回值均为 `.net` 对象.此刻你并不需要知道 `.net` 是什么,你只需要知道他相较于string更容易处理,想必在你学过python的字符串处理之后有了很到位的理解,字符串的处理是十分麻烦的.

cmdlet的命名规则也十分简单易懂,它采用了 动词-名词 的格式.比如获取帮助命令就是 get-help .很好理解对吧?

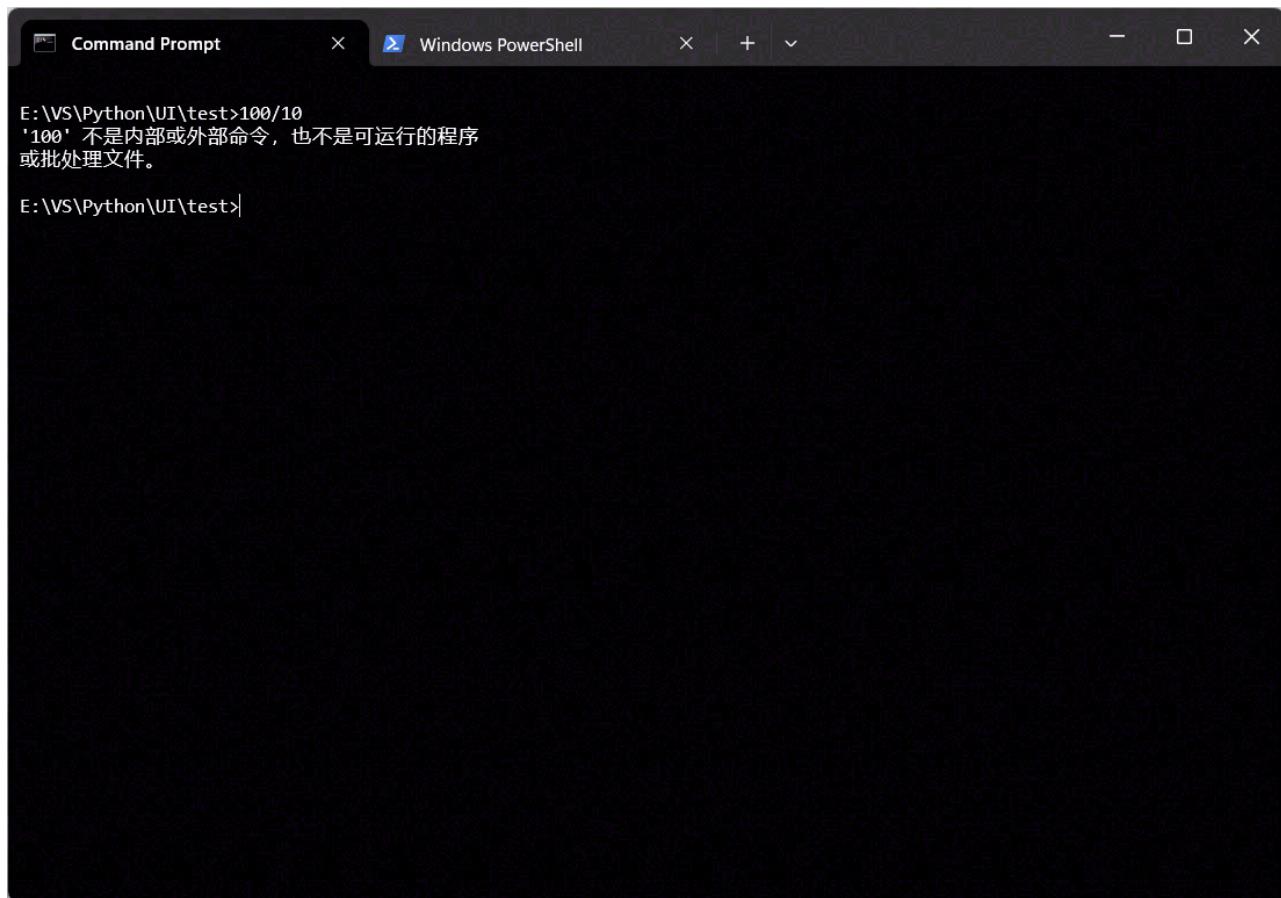
尽管给出的所有cmdlet的规则都是大写,可是实际上并不需要是用大写.

我们来尝试在psh中输入 100\10 .你会发现他是可以正常计算的.



```
PS E:\VS\Python\UI\cmd> 100/10
10
PS E:\VS\Python\UI\cmd> |
```

但是如果是cmd就不行.



```
E:\VS\Python\UI\test>100/10
'100' 不是内部或外部命令, 也不是可运行的程序
或批处理文件。
E:\VS\Python\UI\test>
```

因为cmd发现100/10并不是命令,所以自然无法执行.

通过这个简单的计算,我们就可以看出来psh几乎可以说是一个完整的交互环境了.

类似的优点可以说还有很多很多,比如cmd的文件批处理(.bat)文件不支持if换行.而psh的文件.ps1就完美的支持if换行.不如说.ps1中的if语句已经和正常的高级语言中的if没什么区别了.

这部分可以在附件[批处理指令](#)中的if命令部分非常清晰的看到.

我们看到了如此多的缺点让我们知道,cmd几乎可以说没有一点胜过于psh,那为什么微软依然要保留cmd呢?

原因很简单,并不是微软不想去除cmd,而是不能,因为你永远也不知道全球到底有多少公司的底层使用bat脚本进行维护和运行,你也不知道这些公司给那些后生的公司提供了支持,如果贸然删除cmd,很可能产生不可估计的连锁反应,这样的后果,是微软承担不起的.

这个事情有一个专属名词叫做**技术负债**.这也解释了为什么程序代码会说有很多屎山代码.都是历史遗留问题,人们也无能为力.

可以参考澳大利亚银行花费5年时间耗费7.5亿美金试图更换古老的语言cobol所写的银行系统.想必这个时间成本和经济成本已经高的超出了正常人的预期了,不是吗?

那为什么我接下来仍然选择介绍cmd呢?

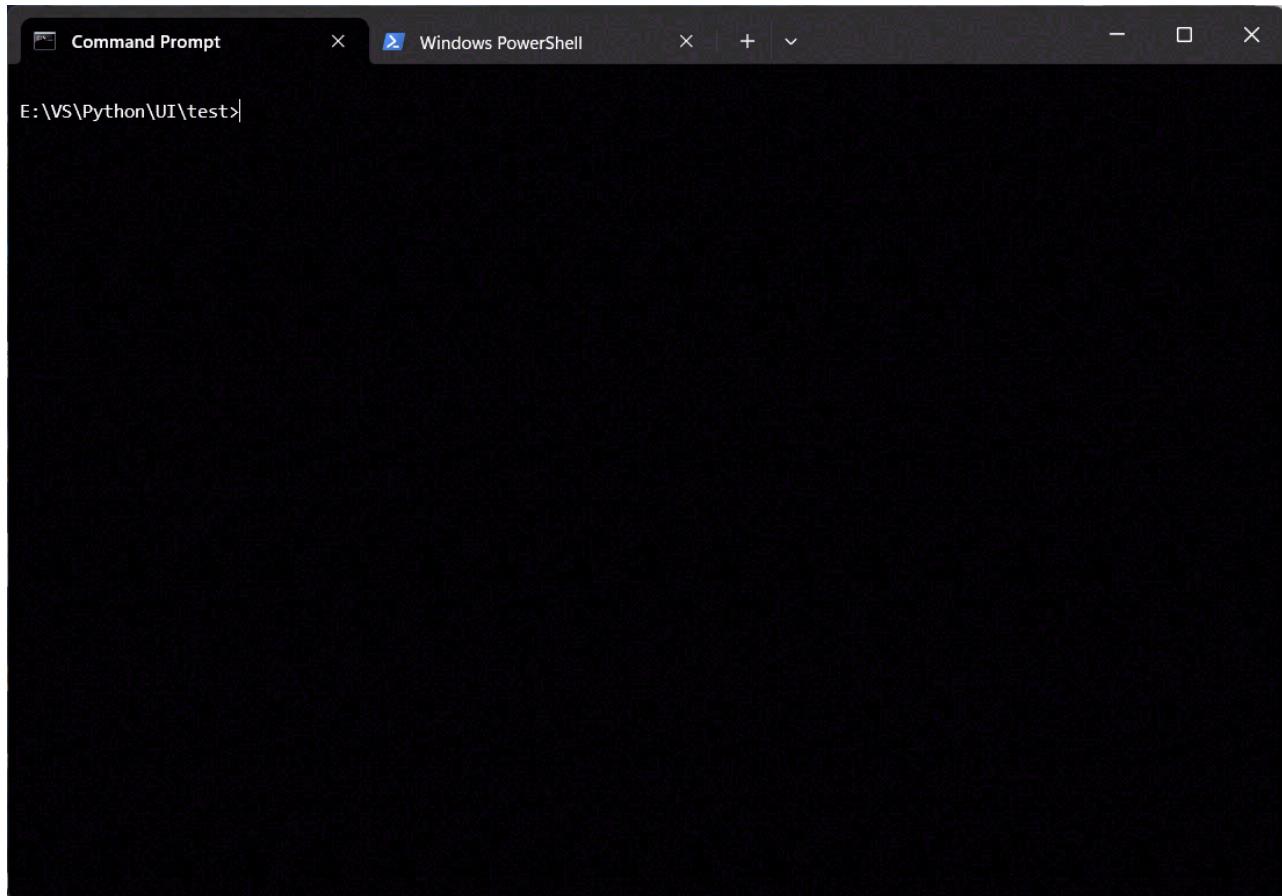
1. cmd用户基本上可以完全无缝过渡到psh,因为大部分windows命令已经被使用别名重现在了psh中.
2. cmd简单,稳定,你可以在任何一台甚至是xp的系统上使用它.最重要的是,它运行**非常非常快!**
3. 因为我之前系统的学过cmd,并记录下了1.5w字的笔记,不偷懒真的太可惜了.

4.2 CMD窗口的理解和配置

那接下来我来简单介绍一下windows常用的命令,更详细的命令可以查看我的附件,这是我专门关于cmd的命令,其中我不仅讲解了简单的文件操作命令,也讲解了很多网络操作的命令和很多系统操作的命令.

4.2.1 联系资源管理器和cmd窗口

cmd窗口和资源管理器类似,接下来我们将他们进行对应.



我们观察到一个cmd窗口提供的信息极其有限,只有一个命令提示符 E:\VS\Python\UI\test> 就没有了,它告诉了我们当前所在的路径,这里就可以对应到我们资源管理器的地址栏部分.除此之外,就没有任何信息了.

极度简单的窗口带来的是极度简单的信息展示,我们需要任何信息都需要我们手动输入指令来获取,这是一件好事,它可以避免我们被过多的无关信息干扰到.

此时我们输入指令 `dir`.有如下输出.

```
1 驱动器 E 中的卷是 Project
2 卷的序列号是 9838-DE48
3
4 E:\VS\Python\UI\test 的目录
5
6 2024/08/06 20:04    <DIR>          .
7 2024/08/06 20:26    <DIR>          ..
8 2024/08/03 10:43    <DIR>          build
9 2024/08/05 08:34    <DIR>          char
10 2024/08/03 10:52      960 clickLog.spec
11 2024/08/03 10:52    <DIR>          dist
12 2024/08/03 09:14      9,966 favicon.ico
13 2024/08/03 00:11    <DIR>          favicon_logosc
14 2023/03/08 23:00      859,483 splash.png
15 2024/08/06 20:04      553 test.py
16                      4 个文件        870,962 字节
17                      6 个目录  5,259,808,768 可用字节
```

这里输出的信息可以和这里联系上.

名称	修改日期	类型	大小
build	2024/8/3 10:43	文件夹	
char	2024/8/5 8:34	文件夹	
dist	2024/8/3 10:52	文件夹	
favicon_logosc	2024/8/3 0:11	文件夹	
ClickLog.spec	2024/8/3 10:52	SPEC 文件	1 KB
favicon.ico	2024/8/3 9:14	ICO 图片文件	10 KB
splash.png	2023/3/8 23:00	PNG 图片文件	840 KB
test.py	2024/8/6 20:04	Python File	1 KB

等等.

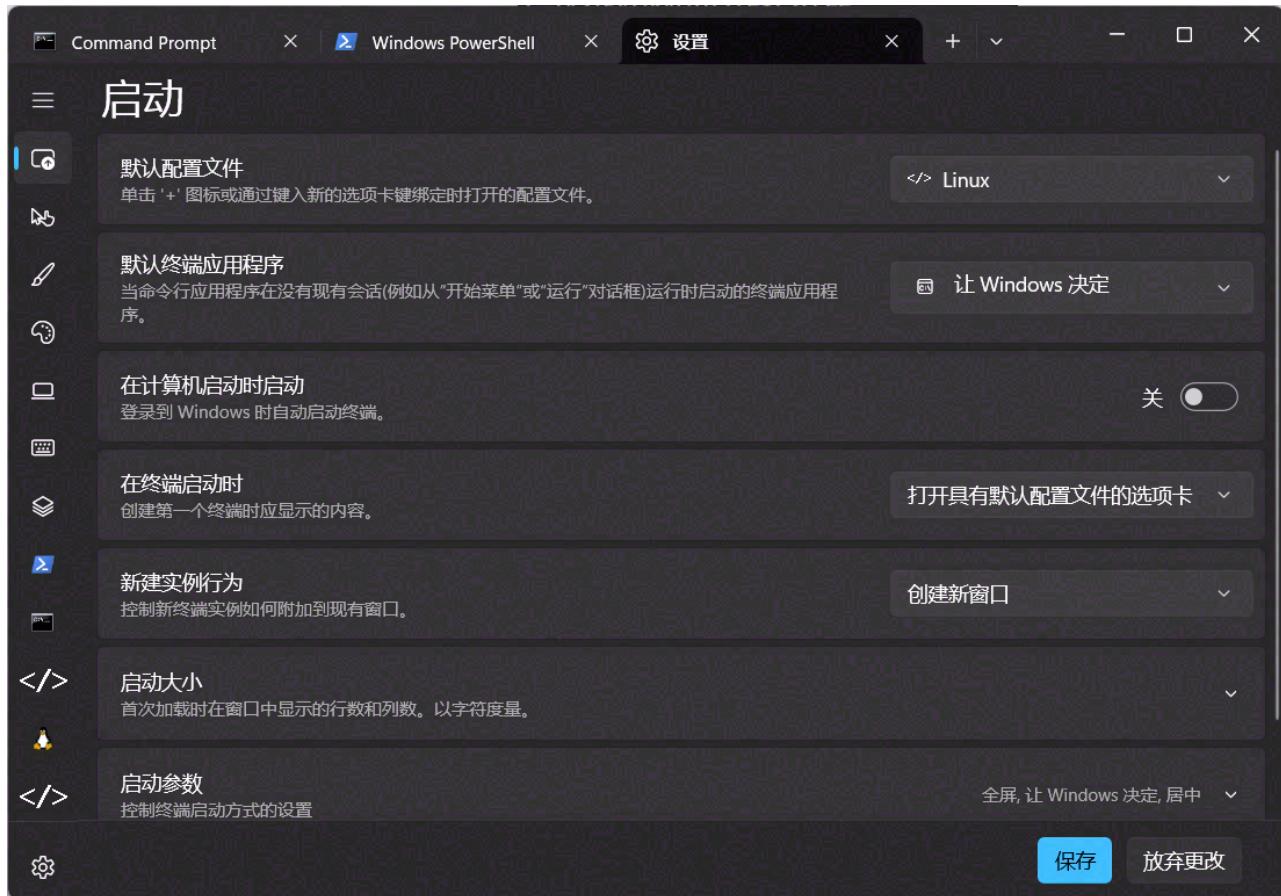
再比如我们使用 `tree` 命令来查看当前文件夹结构.

```
E:\VS\Python\UI\test>tree
卷 Project 的文件夹 PATH 列表
卷序列号为 9838-DE48
E:.
├── build
│   └── ClickLog
│       └── localpycs
├── char
└── dist
└── favicon_logosc

E:\VS\Python\UI\test>
```

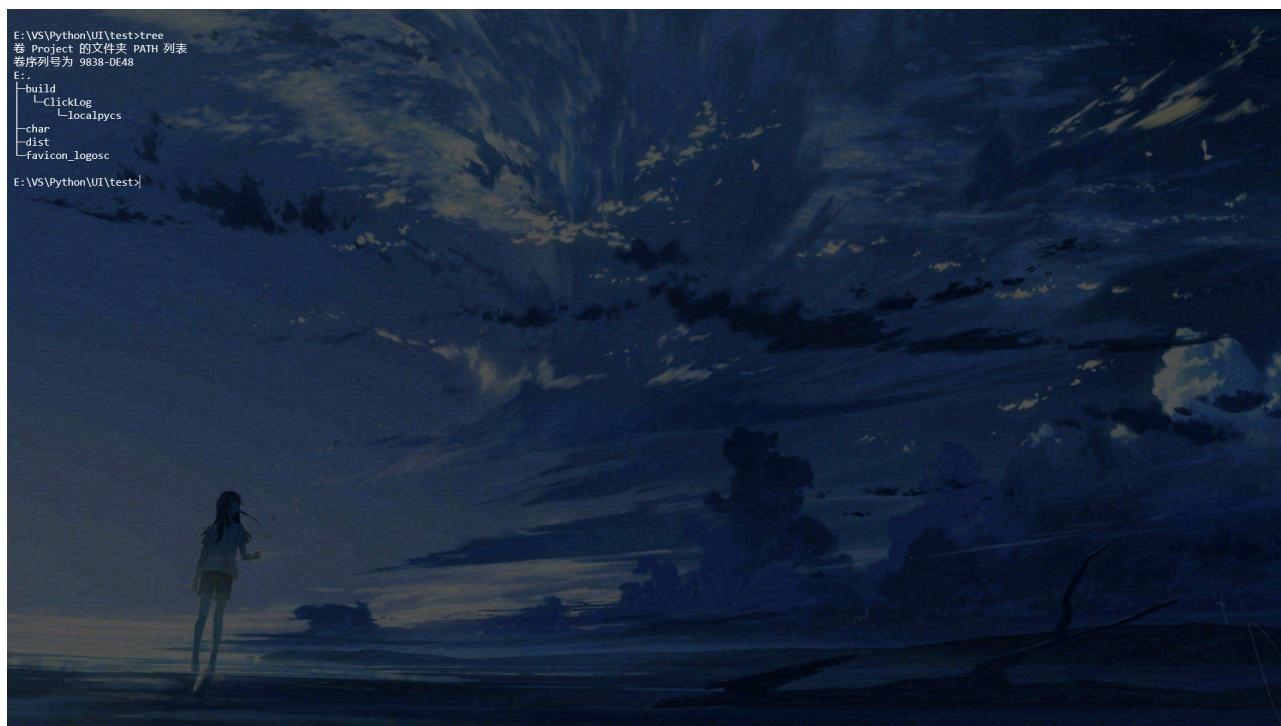
4.2.2 终端的配置

输入 `Ctrl + ,` 打开设置界面



4.2.2.1 简单的美化

其实默认设置就已经够我们使用了,但是考虑到可能有些人觉得默认界面很丑希望更改外观,所以说一下这里可以更改外观.



并不是简单的设置背景这样的,还有更改了代码的显示颜色啊字体啊之类的.以及这个背景可以设置动态壁纸,所以你甚至可以把它当作电脑待机页面.

以及他还有一个除了美观毫无任何作用的显示模式.使用 $\text{Ctrl} + \text{~}$ 打开.



```
E:\VS\Python\UI\test>tree
卷 Project 的文件夹 PATH 列表
卷序列号为 9838-DE48
E:.
├── build
│   └── ClickLog
│       └── localpycs
├── char
└── dist
    └── favicon_logos.c

E:\VS\Python\UI\test>
```

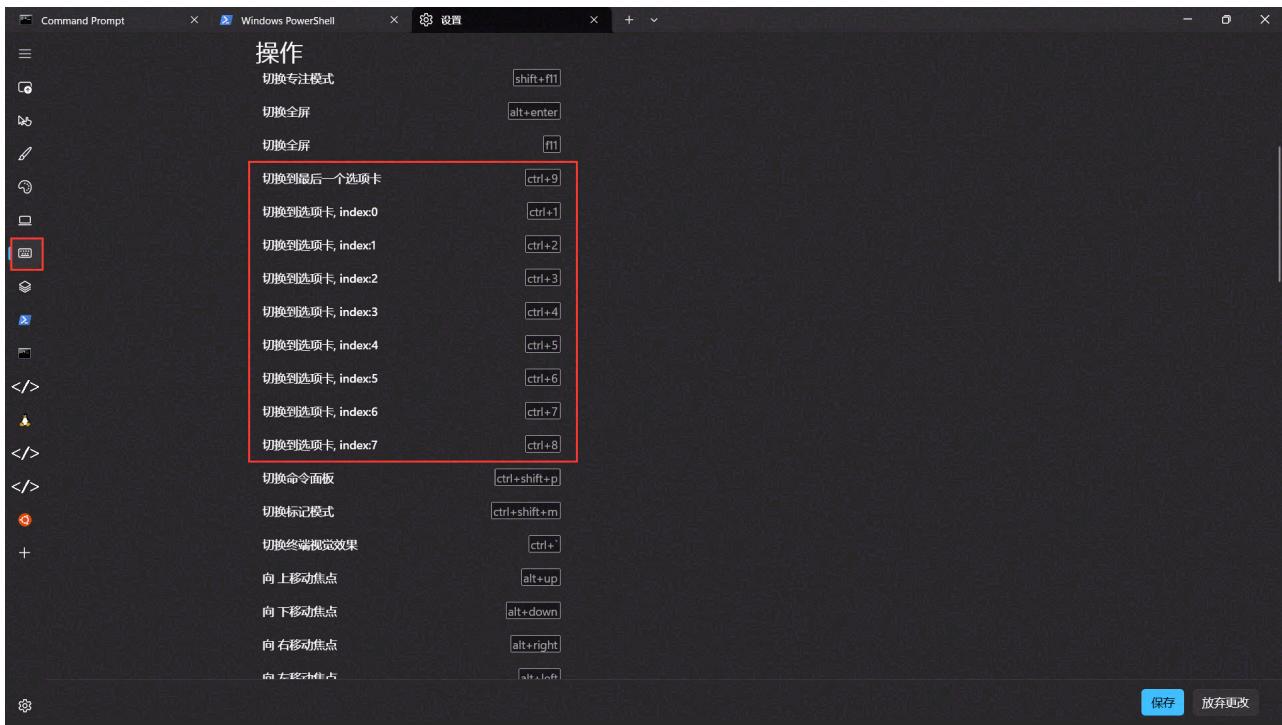
把文字的样式改成上面这个样子.但是我的建议是真的不要开,除了耍帅没有任何作用,甚至还影响你正常使用终端,因为他会影响你有底色的文字显示.



比如这样的底色文字真的会影响人的判断所以建议不要用.

4.2.2.2 快捷键

默认配置其实已经够用了,但是部分快捷键我做了修改.



在操作页面将所有的切换选项卡快捷键从 `Ctrl` + `Alt` + `数字` 更改为 `Ctrl` + `数字`.

因为在浏览器中切换选项卡的快捷键就是 `Ctrl` + `数字`,当然微软这么设计有它的原因(和前置按键的设计思想有关系,但是微软的快捷键设计的依托,还不开放用户自行更改快捷键,所以就不要用它的快捷键了).

在终端中我们并不常用快捷键,大部分功能可以通过指令完成,只有少数的快捷键需要使用.

1. `Ctrl` + `C`

这里并不代表复制,他的意思是停止,终止,用于结束某些长期执行的命令例如我是用 `ping` 命令向bilibili发送数据包,并设置数量为-1,这代表着一直发送直到手动停止.

```
PS E:\VS\Python\UI\cmd> ping www.bilibili.com -n -1

正在 Ping a.w.bilicdn1.com [111.48.57.46] 具有 32 字节的数据
来自 111.48.57.46 的回复: 字节=32 时间=47ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=19ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=37ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=39ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=37ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=38ms TTL=54
来自 111.48.57.46 的回复: 字节=32 时间=37ms TTL=54

111.48.57.46 的 Ping 统计信息:
    数据包: 已发送 = 10, 已接收 = 10, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 19ms, 最长 = 47ms, 平均 = 36ms
Control-C
PS E:\VS\Python\UI\cmd> |
```

观察到上图中最后一个 `control-c`,这代表我用 `Ctrl` + `C` 手动停止了数据包的发送.

2. `Ctrl + L`

清空窗口,非常好用,但是只在psh窗口生效

3. `Ctrl + Backspace`

清空光标到开头(其实是清空到分隔符,分隔符一般是空格,但是没有空格的时候就默认清空到开头)的所有字符.

常用于清空所有输入的字符.

比如 `E:\vs\Python\UI\test>abcdefg` 中,光标在d后面按这个快捷键,就会变成

`E:\vs\Python\UI\test>efg`,如果在g后面按就会变成 `E:\vs\Python\UI\test>.` 在所有字符后面按的时候会清空当前输入的所有字符,这时候效果和按下了 `ESC` 一样.

4. `Ctrl + H`

相当于 `Backspace`.

5. `Ctrl + M`

相当于 `Enter`. 在psh中无效.

6. `Ctrl + 0/-/=`

更改终端字体大小

- 0 默认大小
- 字体变小
- = 字体变大

7. `Ctrl + Shift + 数字`

依照配置文件顺序新建窗口,一般常用两个, `Ctrl + Shift + 1` 新建一个psh窗口, `Ctrl + Shift + 2` 新建要给cmd窗口.

8. `Ctrl + Shift + P`

打开命令窗口,在这里可以快速的执行一些终端命令,实际上在VSCode中也有这个快捷键.

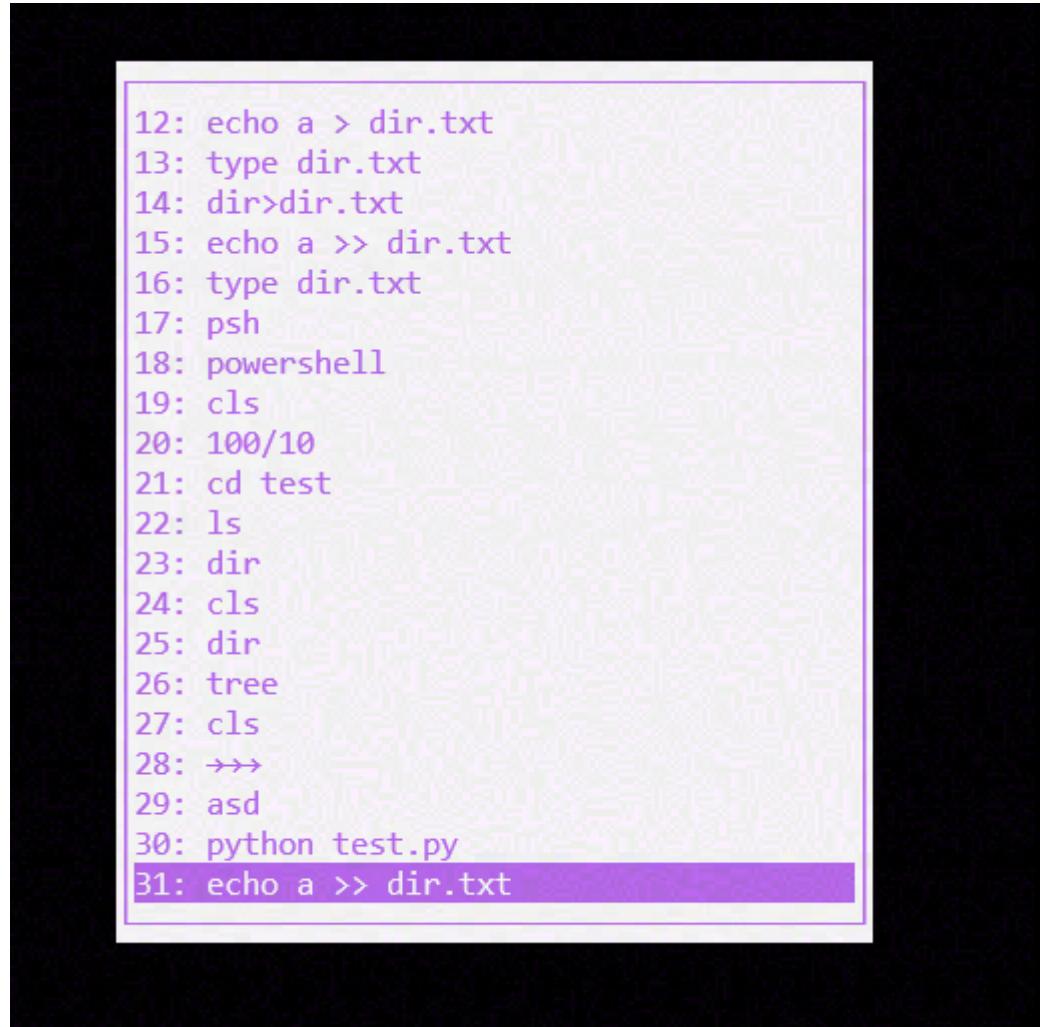
9. `Ctrl + Shift + C`

复制,需要手动选中信息.

剩下是一些在cmd中用得比较多的.

1. `F7`

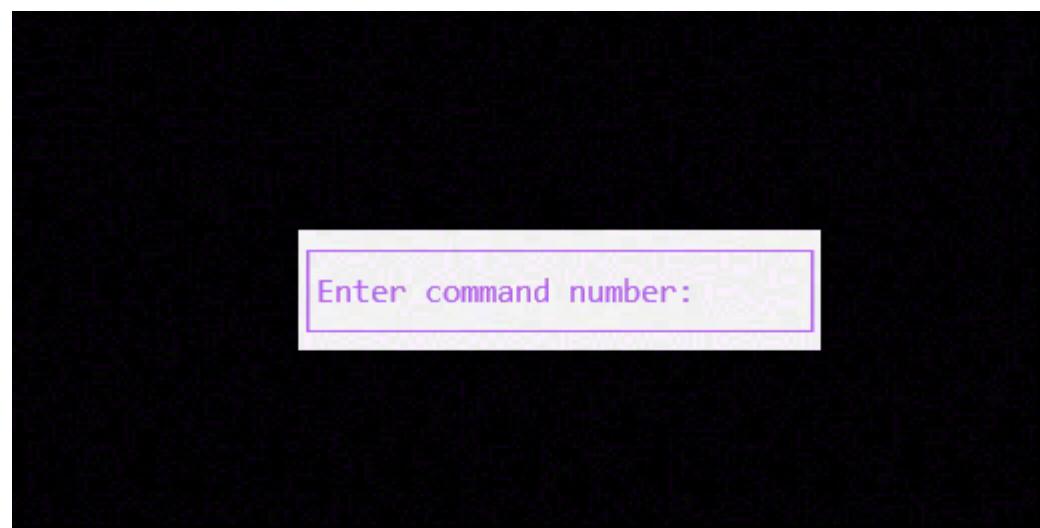
查看命令的历史页面



```
12: echo a > dir.txt
13: type dir.txt
14: dir>dir.txt
15: echo a >> dir.txt
16: type dir.txt
17: psh
18: powershell
19: cls
20: 100/10
21: cd test
22: ls
23: dir
24: cls
25: dir
26: tree
27: cls
28: >>>
29: asd
30: python test.py
31: echo a >> dir.txt
```

上图中31被选中,按键盘上的↑和↓可以上下选择,按下Enter可以执行. ESC关闭页面

2. F9



要求我们输入历史记录上的一个数字他会把这个命令填充到命令提示符后,但是并不执行.

3. ↑和↓

快速输入以前输入过的命令,想要执行上一次执行的命令只需要↑和Enter就好了.

4. Tab 和 Shift + Tab

自动补全和反向自动补全.

这个按键非常非常非常非常非常常用.只要你一天还在写代码,你就一天用到它,它不仅仅是在cmd中,更是在任何你可以写代码的地方,它可以帮助你**自动补全代码**.大大的节省你的时间,比如你写 `im` 然后按下 `Tab`,它会自动帮你补全成为 `import`.

在选择文件的情况下也是非常常用比如 `python test\` 然后你按下 `Tab`,它会自动帮你填充当前目录下的文件和文件夹,如果不是你选择的,你可以多按几下,直到找到你想要的为止,错过了就使用 `Shift + Tab` 来反向遍历.

顺便一提,在cmd中不能自动补全命令和参数,但是在psh中是可以使用 `Tab` 自动补全命令和参数的,这下抛弃cmd的理由日常加一.

其他的就没什么了,剩下一些奇奇怪怪的快捷键,功能也十分稀奇古怪,想要了解自行了解吧,也许在上古cmd时代很常用?

4.3 常用命令

学前准备:

为了方便理解并使用下面命令,所以还请在附件中拷贝cmd文件夹,并放置到合适的位置.

我的cmd文件夹放在了 `E:\VS\Python\UI` 下,所以我所有的命令都是以他为前提.以下所有的命令的地址还请自行修改至自己电脑相对的位置.

检查是否有cmd文件夹.在目录下使用tree并在输出信息中找到cmd文件夹.你应该会得到如下文件夹结构输出.

```
1 E: .
2   └─folderOne
3     |   └─folder
4     |     └─output
5     └─folderTwo
6       |   └─fileFolder
7       |     └─input
8     └─folderZero
9       |   └─zeroCut
10      └─zeroFile
```

3. 路径相关的知识

1. `.` 和 `..` 和 `\` 分别代表了什么

`.` 代表了当前目录.

`..` 代表了上一级目录

`\` 代表了根目录.

2. `/` 和 `/` 哪个用来表示目录.

当然是 `/` 用来表示目录层级了,但是实际上在一些软件的配置文件中为了防止出发转义字符的规则,需要使用 `\` 来表示 `\`.

不过当你在部分软件中使用 `/` 来表示路径的时候也会识别成功,以及在psh中他会自动帮你纠正并正确识别.不过cmd就没有这种功能了.

转义字符:

用于表示不能输入的字符,规则是 `\名称`.使用这种方式可以便捷的完成某些功能.当计算机识别到 `\` 的时候就会认为你使用了转义字符.

最常用的是两个\n用于表示换行,\t表示制表符用于对齐输出.另外还有表示警示的提示音也比较常用,但是我没有用过.

所以\\可以理解为使用\\前缀来转义了\\这个字符.毕竟单独的\\会触发转义字符识别.

还有别的特殊字符输入方式,比如按住Alt不放,在小键盘区依次按下1 2 0 6.你应该会输入这个字符||.

3. 路径是否需要双引号括起来?

双引号并不是必须的,但是如果文件名中存在空格,就需要使用双引号括起来,以防止cmd错误分割命令.

4. .\是必须的吗?

我们知道.是代表了当前目录,所以.\的意思自然就是当前目录下的某个文件.

比如.\out.txt的意思是当前目录下的名叫out的文本文件.那么我直接使用out.txt实际上也可以找到这个文件.

所以.\并不是必须的,但是我一般会写就是了.(在psh中使用tab补全的时候也会帮你写上)

4. dir命令

```
1 | dir [filepath]
```

列出目标路径下的文件和文件夹,如果不写filepath就列出当前目录下的文件和文件夹.

现在我在cmd这个文件夹中,我想要看到folderzero这个文件夹中的内容.我应该输入dir folderzero.

此处输入dir按下空格,输入f按下tab键补全,应该会出现folderOne文件夹,此时再点击tab两次,就会切换到folderZero文件夹.

在psh中,dir命令为Get-ChildItem命令的别名,可以使用Get-Alias别名来查看他的真实命令.也就是说,你在psh中使用dir实际上是在使用Get-ChildItem命令.

值得注意的是,我们观察到在psh中的别名机制中,还有一个需要注意的Get-ChildItem别名,即ls.ls是Linux中的dir命令,以及在Linux中也很常用ll命令来列出当前目录下的文件夹.其中ll也就是ls -l的别名.

考虑到以后可能会接触到linux,所以下面的命令我也会介绍部分linux的知识.但是请注意,附录文件中的命令仅仅在cmd中生效,并不保证在linux中生效.

别名机制

在psh中,为了兼容linux用户和旧的cmd用户,微软提出了别名机制,通过为新的命令起别名,来实现学习成本的降低.具体到用户层面就是cmd的用户可以无缝衔接使用psh,linux用户也基本可以使用常用的基本命令.

例如dir和Get-ChildItem互为别名.

5. cd命令

```
1 | cd [filepath]
```

用于切换路径,值得注意的是仅切换盘符的时候,不能使用cd E:,应该使用E:切换盘符.

现在我在cmd这个文件夹中,使用cd folderzero命令进入folderzero文件夹.使用dir命令列出当前目录下的文件.

结合上面的路径相关知识,可以做到dir ..\..\test在folderzero文件夹中看到test文件夹中的文件.

psh中cd命令的别名为 set-Location,以cd为典型的例子, set-Location 就是体现了psh强大的最好证据,在cmd中,cd仅仅只可以改变路径,但是在psh中 set-Location 不仅可以改变文件路径,还可以改变注册表路径等.虽然我也只是听说没有实际用过,不过都有了注册表编辑器了,正经人不是写脚本,也根本不会用命令行去改吧?

cd命令在linux中可以正常使用.在cmd或者psh中可以使用 cd.. 来切换到上级目录,但是在linux中必须使用 cd .. 来切换.(cd中间有个空格)

6. md命令

```
1 | md [folderpath]
```

用于创建文件夹,值得注意的是, mkdir 和 md 的效果在cmd或者psh中是一样的.

在psh中md命令的别名为 mkdir,这也是我知道的为数不多的不使用cmdlet命名规则的命令.

在linux中,没有md命令,只有 mkdir 命令.

7. move命令

```
1 | move [filepath1] [filepath2]
```

移动命令,也可以当作剪切命令来使用.将文件1移动至文件2.

可以尝试使用cd进入 folderzero 文件夹下,使用 move zeroCut zeroFile 命令将 zeroCut 命令移动至 zeroFile 目录下.

psh中 move 为 Move-Item 的别名.

在linux中应该使用 mv 命令来实现以上效果.

值得注意的是,这条命令还通常用来重命名,使用 move "RenameMe!.txt" "RENAME.txt" 将 folderZero 目录下的 RenameMe!.txt 文件重命名为 RENAME.txt .

关于重命名

尽管在cmd中有一个单独的命令ren或者rename用来重命名,但是我还是建议使用move来进行重命名,因为linux中没有ren这个命令.

8. copy命令

```
1 | copy [filepath1] [filepath1]
```

复制文件命令.

在cmd和psh中实际上还有一个更高级的拷贝命令也就是 xcopy ,还请自行学习.

在linux中使用的命令是 cp ,复制目录还需要加上 -r 参数.

9. 删除命令

删除命令在cmd,psh和linux中总共有三条,分别是 del , rm , rd .

平台\命令	del	rm	rd
cmd	✓		✓
psh	✓	✓	✓
linux		✓	

- del 删除文件
- rm 删除文件
- rd 删除目录

从上用途可以看到为什么cmd中没有 rm ,因为功能重复了.那为什么linux中没有 rd 呢?实际上在Linux中删除目录需要使用 rm -r 来删除,也就是使用了 -r 参数,你可以像为 ls -l 起别名为 ll 那样,将 rm -r 其别名为 rd .

有时候删除文件夹会有提示让你确认是否要删除,这时候他们会让你输入一个 (Y\N) .此时你依照要求输入y 或者n即可,这一步用于确认防止误删除文件的,你也可以在参数中指定不提醒.

10. 创建文件

实际上在cmd中并没有直接用于创建文件的命令,不过我们可以通过别的方式来达到创建文件的目的.

例如 echo abc>>out.txt 这条命令就会创建一个叫做 out 的文本文件(*.txt),它的内容是 abc .单个 > 是覆盖,两个 >> 是追加.

psh中则是使用 new-item 命令来创建文件.

linux中则有一个专门创建文件的命令 touch .

11. 查看文本文件内容

在cmd中一般使用的是 type 命令,例如 type out.txt 你应该可以看到 abc 的输出.

在linux中则需要使用 cat 命令.使用 cat out.txt 可以查看文件内容.实际上linux用还有很多文件内容的命令,比如 head 和 tail 命令.

在psh中 type 和 cat 则都可以使用

截止到现在,基本的命令操作,复制,查看,改变目录,创建,删除,重命名这些功能都已经介绍完毕了,在命令行中还有很多有趣的操作和有意思的命令,这里我就不展开说了,例如被称为管道符的 | 和重定向符 > 这两个符号都是很有用的,仅仅通过以上的介绍并不能很好的理解psh的强大和方便,只有理解了 | 才会发现psh的魅力所在.

以上命令我都仅仅介绍了基本的使用方法,详细的参数和使用规则我就没有详细介绍了,我的附录文件中应该会有常用的参数.

4.4 方便的小技巧

4.4.1 不想使用命令行了,好麻烦

有时候初学者难免会不想使用命令行了,想要打开资源管理器手动操作,这是很正常的事情,可是打开资源管理器再到对应目录,这确实很麻烦,所以不妨使用一条命令来解决 start . 这条命令使用了 start 命令,并传入了路径 . ,也就是当前路径,所以windows就使用资源管理器打开了当前路径.非常好用,不是吗?

start命令的作用是在新窗口中打开些什么,这意味着你可以使用 start www.bing.com 来在浏览器中打开必应,随便什么网址都是可以的.后面可以跟上任何你想要打开的文件.

4.4.2 在psh中使用cmd

还记得我上文说过,其实打开cmd和psh没有区别吗?这是因为我们可以在psh中使用cmd.

我们可以直接在psh中输入cmd并按下回车,就会进入cmd中,在cmd中也是一样的,我们可以输入powershell按下回车进入psh中.

当我们想要退出的时候,可以使用 exit 命令退出.

4.4.3 帮助界面

想要学好命令,帮助界面是必不可少的.

不同平台的获取帮助方式不同,这里我总结了常见的几种获取帮助的方式.

cmd中使用 `/?` 参数来获取对应命令的帮助.例如 `xcopy /?`.

```
E:\VS\Python\UI\cmd\folderTwo>xcopy /?
复制文件和目录树。

XCOPY 源 [destination] [/A | /M] [/D[:date]] [/P] [/S [/E]] [/V] [/W]
          [/C] [/I] [/N] [/Q] [/F] [/L] [/G] [/H] [/R] [/T]
          [/U] [/K] [/N] [/O] [/X] [/Y] [/Z] [/B] [/J]
          [/EXCLUDE:file1[+file2][+file3]... ] [/COMPRESS]

source      指定要复制的文件。
destination 指定新文件的位置和/或名称。
/A          仅复制具有存档属性的文件。
           不会更改属性。
/M          仅复制具有存档属性集的文件。
           关闭存档属性。
/D:m-d-y   复制指定日期或之后更改的文件。
           如果未提供日期，仅复制其源时间
           比目标时间晚的文件。
/EXCLUDE:file1[+file2][+file3]...
           指定包含字符串的文件列表。每个字符串
           应在文件中单独一行显示。当有文件
           字符串与要复制的文件的绝对路径的任何部分
           四时时，将从文件中排除该文件。
           如果指定 \obj\ 或 \obj\*，字符串将分别
           指移到 \obj\ 目录下的所有文件或具有 .obj 扩展名的
           所有文件。
/P          在创建每个目标文件之前提示你。
/S          复制目录和子目录(空目录除外)。
/E          复制目录和子目录(包括空目录)。
           与 /S /E 相同。可用于修改 /T。
/V          验证每个新文件的大小。
/W          提示你在复制前按一个键。
/C          即使发生错误，也继续复制。
/I          如果目标不存在且复制多个文件，则假定目标必须存在目录。
           如果目标不存在且复制单个指定文件，则假定目标必须是文件。
/-I         在复制时不显示文件名。
/Q          在复制时显示完整的源文件名和目标文件名。
/F          显示要复制的文件。
/L          允许将加密文件复制到
           不支持加密的目录。
/H          读取帮助文件或系统文件。
/R          覆盖只读文件。
/T          创建目录结构，但不复制文件。不
           包括空目录或子目录。/T /E 包含
           空目录和子目录。
/U          复制属性。普通 Xcopy 将重置只读属性。
/K          使用生成的短名粘贴。
/O          复制文件所有权限和 ACL 信息。
/O          复制文件审核设置(隐含 /O)。
/X          禁止提示确认是否要覆盖
/Y          禁止提示确认是否要覆盖
```

psh中可以使用 `get-help` 命令来获取对应命令的帮助信息.例如 `get-help Get-childitem`.

```
PS E:\VS\Python\UI\cmd\folderTwo> get-help Get-childitem

PS E:\VS\Python\UI\cmd\folderTwo> Get-Help Get-childitem

名称
    Get-ChildItem

语法
    Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>]  [<CommonParameters>]

    Get-ChildItem [[-Filter] <string>]  [<CommonParameters>]

别名
    gci
    ls
    dir

备注
    Get-Help 在此计算机上找不到该 cmdlet 的帮助文件。它仅显示部分帮助。
    -- 若要下载并安装包含此 cmdlet 的模块的帮助文件，请使用 Update-Help。
    -- 若要联机查看此 cmdlet 的帮助主题，请键入："Get-Help Get-ChildItem -Online" 或
       转到 https://go.microsoft.com/fwlink/?LinkID=113308。
```

linux中可以使用 `-h` 或者 `--help` 参数来获取对应命令的帮助.例如 `mkdir -h`.

```

└─(dsgh㉿DSGHBC)-[~]
$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
-m, --mode=MODE    set file mode (as in chmod), not a=rwx - umask
-p, --parents      no error if existing, make parent directories as needed,
                   with their file modes unaffected by any -m option.
-v, --verbose      print a message for each created directory
-Z                 set SELinux security context of each created directory
                   to the default type
--context[=CTX]   like -Z, or if CTX is specified then set the SELinux
                   or SMACK security context to CTX
--help            display this help and exit
--version         output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'

```

大部分工具也都会提供 `--help` 或者 `-h` 参数用于获取帮助.

```

PS E:\VS\Python\UI\cmd\folderTwo> python --help
usage: D:\Program Files (x86)\Python3.10.2\python.exe [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-b : issue warnings about str(bytes_instance), str(bytearray_instance)
     and comparing bytes/bytarray with str. (-bb: issue errors)
-B : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x
-c cmd : program passed in as string (terminates option list)
-d : turn on parser debugging output (for experts only, only works on
     debug builds); also PYTHONDEBUG=x
-E : ignore PYTHON* environment variables (such as PYTHONPATH)
-h : print this help message and exit (also --help)
-i : inspect interactively after running script; forces a prompt even
     if stdin does not appear to be a terminal; also PYTHONINSPECT=x
-I : isolate Python from the user's environment (implies -E and -s)
-m mod : run library module as a script (terminates option list)
-o : remove assert and __debug__-dependent statements; add .opt-1 before
     .pyc extension; also PYTHONOPTIMIZE=x
-OO : do .c changes and also discard docstrings; add .opt-2 before
     .pyc extension
-q : don't print version and copyright messages on interactive startup
-r : don't add user site directory to sys.path; also PYTHONUSER_SITE
-S : don't imply 'import site' on initialization
-u : force the stdout and stderr streams to be unbuffered;
     this option has no effect on stdin; also PYTHONUNBUFFERED=x
-v : verbose (trace import statements); also PYTHONVERBOSE=x
     can be supplied multiple times to increase verbosity
-V : print the Python version number and exit (also --version)
     when given twice, print more information about the build
-W arg : warning control; arg is action:message:category:module:lineno
     also PYTHONWARNINGS=arg
-x : skip first line of source, allowing use of non-Unix forms of #!cmd
-X opt : set implementation-specific option. The following options are available:

-X faulthandler: enable faulthandler
-X showrefcount: output the total reference count and number of used
     memory blocks when the program finishes or after each statement in the
     interactive interpreter. This only works on debug builds
-X tracemalloc: tracing Python memory allocations using the
     tracemalloc module. By default, only the most recent frame is stored in a
     traceback of a trace. Use -X tracemalloc=NFRAME to start tracing with a
     traceback limit of NFRAME frames
-X importtime: show how long each import takes. It shows module name,
     cumulative time (including nested imports) and self time (excluding
     nested imports). Note that its output may be broken in multi-threaded
     application. Typical usage is python3 -X importtime -c 'import asyncio'
-X dev: enable CPython's "development mode", introducing additional runtime
     checks which are too expensive to be enabled by default. Effect of the
     developer mode:
     * Add default warning filter, as -W default
     * Install debug hooks on memory allocators: see the PyMem_SetupDebugHooks() C function
     * Enable the faulthandler module to dump the Python traceback on a crash
     * Enable asyncio debug mode

```

```

PS E:\VS\Python\UI\cmd\folderTwo> pyinstaller --help
usage: pyinstaller [-h] [-v] [-D] [-F] [-specpath DIR] [-n NAME] [-contents-directory CONTENTS_DIRECTORY] [-add-data SOURCE:DEST] [-add-binary SOURCE:DEST] [-p DIR] [-hidden-import MODULENAME]
                   [--collect-submodules MODULENAME] [--collect-data MODULENAME] [--collect-binaries MODULENAME] [-collect-all MODULENAME] [-copy-metadata PACKAGENAME] [-recursive-copy-metadata PACKAGENAME]
                   [-additional-hooks-dir HOOKSPATH] [-runtime-hook RUNTIME_HOOKS] [-exclude-module EXCLUDES] [-splash IMAGE_FILE] [-d (all,imports,bootloader,noarchive)] [-optimize LEVEL]
                   [-python-option PYTHON_OPTION] [-s] [-noupx] [-upx-exclude FILE] [-c] [-w] [-hide-console (hide-late,minimize-late,hide-early,minimize-early)]
                   [-i <FILE.ico or FILE.exe.ID or FILE.icns or Image or "NONE"] [-disable-windows-traceback] [-version-file FILE] [-m <FILE or XML>] [-r RESOURCE] [-uac-admin] [-uac-uiaccess]
                   [-argv-emulation] [-osx-bundle-identifier BUNDLE_IDENTIFIER] [-target-architecture ARCH] [-codesign-identity IDENTITY] [-osx entitlements-file FILENAME] [-runtime-tmpdir PATH]
                   [-bootloader-ignore-signals] [--distpath DIR] [--workpath WORKPATH] [-y] [-upx-dir UPX_DIR] [-clean] [-log-level LEVEL]
scriptname [scriptname ...]

positional arguments:
  scriptname           Name of scriptfiles to be processed or exactly one .spec file. If a .spec file is specified, most options are unnecessary and are ignored.

options:
  -h, --help          show this help message and exit
  -v, --version       Show program version info and exit.
  --distpath DIR      Where to put the bundled app (default: ./dist)
  --workpath WORKPATH Where to put all the temporary work files, .log, .pyz and etc. (default: ./build)
  -y, --noconfirm     Replace output directory (default: SPECPATH\dist\SPECNAME) without asking for confirmation
  --upx-dir UPX_DIR   Path to UPX utility (default: search the execution path)
  --clean             Clean PyInstaller cache and remove temporary files before building.
  --log-level LEVEL   Amount of detail in build-time console messages. LEVEL may be one of TRACE, DEBUG, INFO, WARN, DEPRECATION, ERROR, FATAL (default: INFO). Also settable via and overrides the PYI_LOG_LEVEL environment variable.

What to generate:
  -D, --onedir        Create a one-folder bundle containing an executable (default)
  -F, --onefile       Create a one-file bundled executable
  --specpath DIR      Folder to store the generated spec file (default: current directory)
  -n NAME, --name NAME Name to assign to the bundled app and spec file (default: first script's basename)
  --contents-directory CONTENTS_DIRECTORY
                      For onedir builds only, specify the name of the directory in which all supporting files (i.e. everything except the executable itself) will be placed in. Use "." to re-enable old onedir layout without contents directory.

What to bundle, where to search:
  --add-data SOURCE:DEST
                      Additional data files or directories containing data files to be added to the application. The argument value should be in form of "source:dest_dir", where source is the path to file (or directory) to be collected, dest_dir is the destination directory relative to the top-level application directory, and both paths are separated by a colon (:). To put a file in the top-level application directory, use . as a dest_dir. This option can be used multiple times.
  --add-binary SOURCE:DEST
                      Additional binary files to be added to the executable. See the ``--add-data`` option for the format. This option can be used multiple times.
  -p DIR, --paths DIR A path to search for imports (like using PYTHONPATH). Multiple paths are allowed, separated by `';'`, or use this option multiple times. Equivalent to supplying the ``pathex`` argument in the spec file.
  --hidden-import MODULENAME, --hiddenimport MODULENAME
                      Name an import not visible in the code of the script(s). This option can be used multiple times.
  --collect-submodules MODULENAME
                      Collect all submodules from the specified package or module. This option can be used multiple times.
  --collect-data MODULENAME
                      Collect all data from the specified package or module. This option can be used multiple times.
  --collect-binaries MODULENAME
                      Collect all binaries from the specified package or module. This option can be used multiple times.
  --collect-all MODULENAME
                      Collect all submodules, data files, and binaries from the specified package or module. This option can be used multiple times.


```

linux中还有 `man` 命令可以用于获取命令的详细帮助,对初学者来说是很大的帮助.

直接在psh中输入 `help` 或者 `man` 也可以看到关于psh的帮助信息.

主题
Windows PowerShell 帮助系统
简短说明
显示有关 Windows PowerShell 的 cmdlet 及概念的帮助。
详细说明
Windows PowerShell 帮助介绍了 Windows PowerShell 的 cmdlet、函数、脚本及模块，并解释了 Windows PowerShell 语言的元素等概念。
Windows PowerShell 中不包含帮助文件，但你可以联机参阅帮助主题，或使用 Update-Help cmdlet 将帮助文件下载到你的计算机中，然后在命令行中使用 Get-Help cmdlet 来显示帮助主题。
你也可以使用 Update-Help cmdlet 在该网站发布了更新的帮助文件时下载它们。这样，你的本地帮助内容便永远都不会过时。
如果没有帮助文件，Get-Help 会显示自动生成的有关 cmdlet、函数及脚本的帮助。
联机帮助
你可以在 TechNet 库中找到有关 Windows PowerShell 的联机帮助，网址为 <http://go.microsoft.com/fwlink/?LinkID=108518>。
若要打开有关 cmdlet 或函数的联机帮助，请键入：
Get-Help <cmdlet-name> -Online
UPDATE-HELP
若要下载帮助文件并将其安装在计算机上，请执行以下步骤：
1. 使用“以管理员身份运行”选项启动 Windows PowerShell。
2. 键入：
Update-Help
安装了帮助文件之后，你便可以使用 Get-Help cmdlet 来显示帮助主题。你也可以使用 update-Help cmdlet 来下载更新的帮助文件，让本地帮助文件始终保持为最新。
有关 Update-Help cmdlet 的详细信息，请键入：
Get-Help Update-Help -Online
或转至：<http://go.microsoft.com/fwlink/?LinkID=210614>

所以我们需要确信一点,软件写出来是为了人去使用的,那么如果人不会用,这个软件就没有意义,所以作者一定会以某种方式去告诉我们怎么去使用,无论是上面说的使用参数打开对应帮助信息,还是附加一个 `README.md` 的帮助文档,亦或者是窗口上 `关于->帮助` 界面的几个网址,或者一些约定俗成的习惯,都会帮助我们学习如何使用这个工具.

4.4.4 一些约定俗成的小习惯

在命令行界面中,我们一般默认 `-` 开头表示详细参数信息,而使用 `-` 来表示简化参数信息.上文中提到的帮助信息或者版本介绍正是最好的例子.

关于参数界面,实际上并没有明确的规定,尽管我觉得有很多东西需要一个明确的标准,但是实际上并没有.但是一般来说 `[]` 表示的可选信息, `<>` 表示的必选信息.使用 `|` 分割表示任选其一.但是这些规则并不是所有人都遵守的,我甚至见过一些文档中上下使用的这些规定都会有冲突的现象出现,所以仅仅只需要有个概念就好.

E:\VS\Python\UI\cmd\folderTwo>echo /?
显示消息，或者启用或关闭命令人回显。

ECHO [ON | OFF]
ECHO [message]

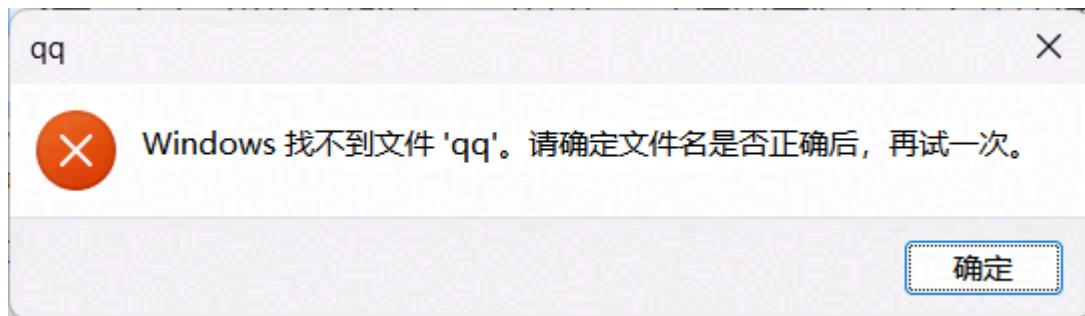
若要显示当前回显设置，请键入不带参数的 ECHO。

这条帮助信息中就给了我们一个例子,对于第一个语法结构,我们需要任选 ON 和 OFF 中的一个.

4.4.5 环境变量

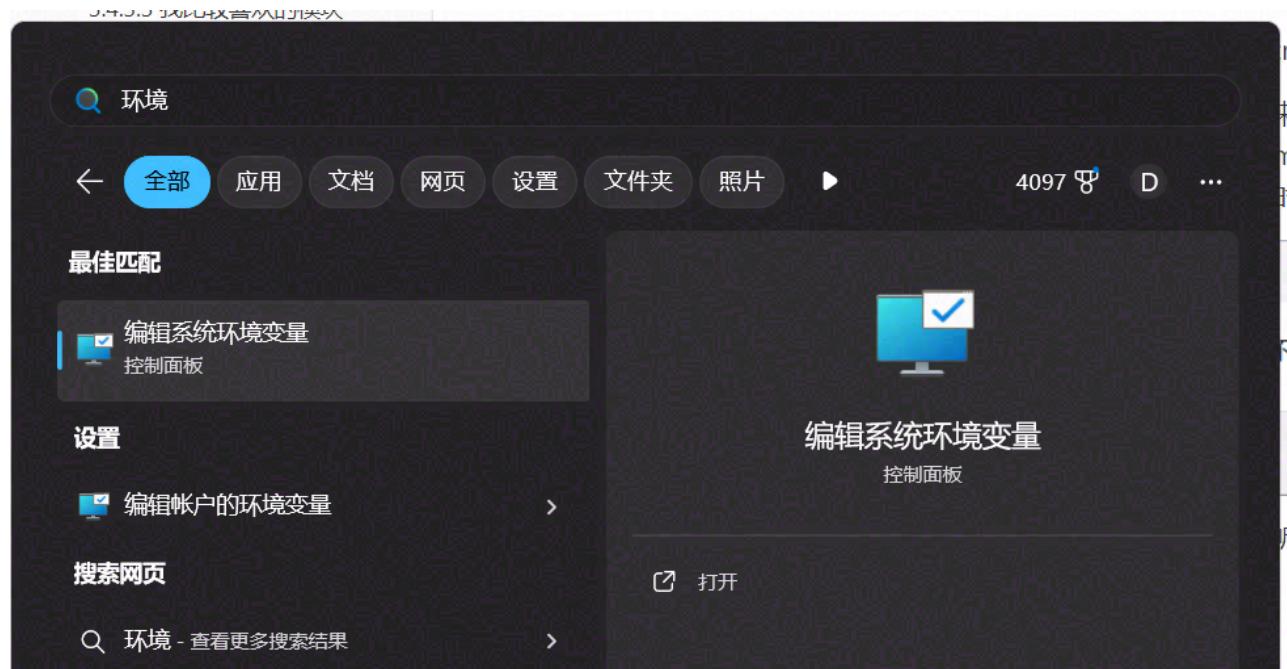
我们在上文中提到了在psh中可以通过输入cmd打开cmd,这里实际上离不开环境变量的功劳.

环境变量决定了**搜索的优先级**.当我们使用 **Win + R** 来打开 运行 窗口的时候,我们输入了cmd,那么程序就会执行 cmd,为什么呢?难道是windows在系统中找遍了文件,最后找到了一个叫做cmd的程序执行了它吗?肯定不是,因为无法避免有重复命名的现象出现,更何况,如果你没有配置环境变量,那么当你在 运行 窗口中输入QQ的时候,他大概率会给你下面的这个输出.



那为什么我们输入 python 的时候却可以打开python呢?这正是因为配置了环境变量.想必在安装的时候应该会看到环境变量这个字样吧.

使用 **Win + Q** 打开搜索界面,输入 环境 .我们可以看到如下输入.(实际上输入环境的拼音也是可以的).



我们打开这个选项,进入如下页面.



再点击最下面的环境变量选项,就可以配置环境变量了.



在上面的这个窗口中,我们需要关注的是名为**Path**的环境变量,你可以配置用户变量,也可以配置系统变量,对于仅仅只有一个用户的个人电脑来说,这没有区别.

用户变量的作用是你做出的修改都只会在这个账户上生效,当你在这台电脑上切换了账户,那么这些设置就没有作用了.

系统变量的作用就是你做出的修改会在这个系统上生效,所以无论你是否切换账户,他都会生效.为了方便起见,一般各个程序都会直接设置环境变量在系统上,不过有些程序会很贴心的在安装的时候让用户个人选择.

编辑环境变量

X

```
D:\Program Files (x86)\Python\3.10.2\Scripts\  
D:\Program Files (x86)\Python\3.10.2\  
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps  
%USERPROFILE%\dotnet\tools  
%IntelliJ IDEA Community Edition%  
E:\Program\MinGW\bin
```

新建(N)

编辑(E)

浏览(B)...

删除(D)

上移(U)

下移(O)

编辑文本(T)...

确定

取消

我们双击进入Path中,可以看到如上内容,这些就是我的用户变量,

上面就有我的python安装目录.在这个目录中,我们能看到python.exe这个程序.

名称	修改日期	类型	大小
DLLs	2022/3/12 22:15	文件夹	
Doc	2022/3/12 22:15	文件夹	
include	2022/3/12 22:15	文件夹	
Lib	2022/3/12 22:15	文件夹	
libs	2022/3/12 22:15	文件夹	
Scripts	2024/8/9 0:04	文件夹	
tcl	2022/3/12 22:15	文件夹	
Tools	2022/3/12 22:15	文件夹	
LICENSE.txt	2022/1/17 14:26	TXT 文件	32 KB
NEWS.txt	2022/1/17 14:27	TXT 文件	1,203 KB
out	2024/8/3 14:55	文件	1 KB
out.txt	2024/8/3 14:22	TXT 文件	0 KB
python.exe	2022/1/17 14:26	应用程序	97 KB
python3.dll	2022/1/17 14:26	应用程序扩展	61 KB
python310.dll	2022/1/17 14:26	应用程序扩展	4,350 KB
pythonw.exe	2022/1/17 14:26	应用程序	96 KB
test.txt	2024/8/3 14:38	TXT 文件	1 KB
vcruntime140.dll	2022/1/17 14:26	应用程序扩展	95 KB
vcruntime140_1.dll	2022/1/17 14:26	应用程序扩展	37 KB

所以当我们输入python时,他就会检索到这个目录下,并执行这个目录中的python.exe文件.

以上是我的个人用户配置,我们同样可以在系统变量配置中找到如下需要关注的配置.

```
%INTEL_DEV_REDIST%redist\intel64\compiler
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
```

将这个路径复制 %SystemRoot%\system32 到资源管理器的地址栏中,我们可以找到 cmd 文件.

名称	修改日期	类型	大小
CloudExperienceHost.dll	2024/7/18 8:13	应用程序扩展	594 KB
CloudExperienceHostBroker.dll	2024/6/14 20:33	应用程序扩展	438 KB
CloudExperienceHostBroker.exe	2024/5/18 2:26	应用程序	94 KB
CloudExperienceHostCommon.dll	2024/7/18 8:13	应用程序扩展	1,354 KB
CloudExperienceHostRedirection.dll	2024/6/14 20:33	应用程序扩展	208 KB
CloudExperienceHostUser.dll	2024/6/14 20:32	应用程序扩展	286 KB
cloudidsvc.dll	2024/5/18 2:23	应用程序扩展	128 KB
CloudIdWxhExtension.dll	2024/6/14 20:32	应用程序扩展	298 KB
CloudNotifications.exe	2024/6/14 20:32	应用程序	107 KB
CloudRecoveryDownloadTool.dll	2022/5/7 13:20	应用程序扩展	3,233 KB
CloudRestoreLauncher.dll	2024/7/18 8:13	应用程序扩展	1,512 KB
clrhost.dll	2022/5/7 13:20	应用程序扩展	36 KB
clusapi.dll	2024/6/14 20:35	应用程序扩展	1,212 KB
cmcfg32.dll	2022/5/7 13:19	应用程序扩展	60 KB
cmd.exe	2024/6/14 20:33	应用程序	316 KB
cmdext.dll	2024/6/14 20:33	应用程序扩展	68 KB
cmdial32.dll	2022/5/7 13:19	应用程序扩展	556 KB

环境变量还有很多有趣的作用,但是限于篇幅原因,我不在赘述什么费马猜想.

4.5 Markdown文档

我在上文中反复提到了 README.md 文件,这个后缀 *.md 代表的就是markdown文档.

markdown是一种标记性语言.你所看到的这个文本就是我是用markdown写的,自从我接触到了md文档之后,我就再也没有打开过word文档了(bushi).

md文档的优点不必多说,语法简单,仅仅只需要5分钟就可以全部学完,扩展性高(支持内部插入HTML,CSS).文本体积小.

我们知道docx使用的是xml文本的方式来记录格式.而所谓的 *.docx 其实质是一个压缩包.你可以修改任何一篇docx的文件后缀为rar直接使用压缩包软件打开.

名称	大小	压缩后大小	类型	修改时间	CRC32
..			文件夹		
word	1,314,228	76,332	文件夹	2012/7/2 9:52	
docProps	1,797	998	文件夹	2012/7/2 9:52	
_rels	737	253	文件夹	2012/7/2 9:52	
[Content_Types].xml	1,433	367	XML文件	2012/7/2 9:52	A64233C1

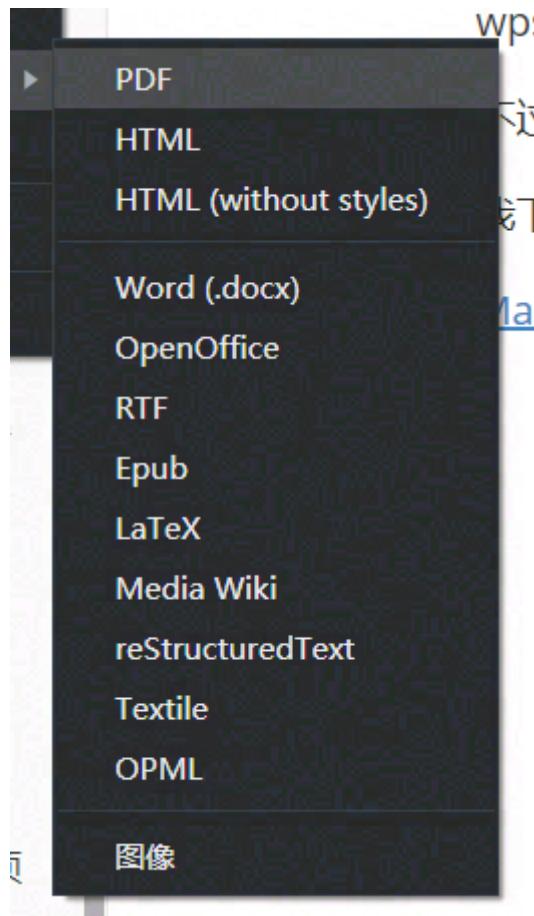
所以理所当然,docx的文件体积会比md文件大得多的多,毕竟md文档就仅仅是txt文档,你甚至可以使用系统自带的记事本打开.这一点和 *.trf 文件很像,不过 *.trf 使用的是富文本.

你可以看到上文中的超链接,数学公式,代码块,按键图标,图片,表格,流程图或者思维导图等等,都是使用了md的语法进行绘制.

它的优点是效率很高,在使用md文档的时候,我其实没有拿出鼠标来,因为所有的格式设置都用不到鼠标,减少了手在键盘和鼠标之间的移动,自然效率就上来了.

那么当然他也有缺点,那就是必须要一个渲染器才可以显示出他的样式,说白了就是需要一个markdown编辑器.就好像你没有浏览器打开*html文件显示的不过是一串代码,只有使用浏览器解析了html代码才会显示正常的信息一样.不过这也并不是什么缺点,毕竟你可以安装wps,不可能不愿意安装十几兆的markdown编辑器不是吗?它体积小,效率高,干净整洁,我实在是想不出不选择他的理由.

不过当你看到这篇文档的时候大概率并不是所谓的md文档,而是*.pdf文件,这也正是他的优点,很方便的转化为各种格式.



这简直太方便了,不是吗?

还请看一看我的这部分的md文档的源码.

3200

环境变量还有很多有趣的作用,但是限于篇幅原因,我不在赘述<_{sub>什么费马猜想}.</sub>

4.5 Markdown文档

我在上文中反复提到了`README.md`文件,这个后缀`*.md`代表的就是markdown文档.

markdown是一种标记性语言.你所看到的这个文本就是我是用markdown写的,自从我接触到了md文档之后,我就再也没有打开过word文档了(bushi).

md文档的优点不必多说,语法简单,仅仅只需要5分钟就可以全部学完,扩展性高(支持内部插入HTML,CSS).文本体积小.

3210

我们知道docx使用的是xml文本的方式来记录格式.而所谓的`*.docx`其实本质是一个压缩包.你可以修改任何一篇docx的文件后缀为rar直接使用压缩包软件打开.

以及我的md文档编辑器的界面.

The screenshot shows a user interface for a Markdown editor. On the left is a sidebar with a tree view of the document structure. The main content area contains text and some code snippets. The text discusses the advantages of Markdown over Word documents and provides a link to the official Markdown documentation.

我在上文中反复提到了`README.md`文件,这个后缀`*.md`代表的就是markdown文档.
markdown是一种标记性语言.你所看到的这个文本就是我是用markdown写的,自从我接触到了md文档之后,我就再也没有打开过word文档了(bushi).
md文档的优点不必多说,语法简单,仅仅只需要5分钟就可以全部学完,扩展性高(支持内部插入HTML,CSS).文本体积小.
我们知道docx使用的是xml文本的方式来记录格式,而所谓的`*.docx`其实本质是一个压缩包.你可以修改任何一篇docx的文件后缀为rar直接使用压缩包软件打开.

以我的md文档编辑器的界面.
|
3210
我们知道docx使用的是xml文本的方式来记录格式,而所谓的`*.docx`其实本质是一个压缩包.你可以修改任何一篇docx的文件后缀为rar直接使用压缩包软件打开.

以我的md文档编辑器的界面.
|
37163 词

真的干净的没有什么可以说的不是吗?

我下面给出我推荐的md文档教程.

[Markdown 官方教程.](#)

我使用的编辑器是 Typora 的测试版本,额对,他的最新版都已经开始收费了.不过他的古老开发版我用到现在4年多了也没有出过任何问题,哪怕这篇文章已经写到了3.7w字,也依然流畅运行,只要不突然编辑上面1w字处的信息就行,不过即使你去编辑那里的信息,也是可以正常使用的,只不过有点卡而已.然后切换源码模式则不会有任何卡顿出现.

以及我看了他后面的更新,都加了一些无关紧要的功能,所以我觉得这个已经够用了.这个测试版做到了需要什么有什么,不需要的功能是一个没有.

[Typora测试版下载链接.](#)

4.6 如何自学终端的命令?

在cmd中,我们输入`help`我们可以看到所有的命令,不知道你是否会感叹命令之多呢?实际上,在我看来命令的数量,是永远学习不完的.如果仅仅是系统自带的命令,或许花费些时间总会学习完,但是如果考虑到种种工具的命令,想必此生我们也学习不完所有的命令.所以,命令究竟要怎么学呢?

我自己的答案是按需学习.我只会学习我用的到的命令,并把它记住,如果我将这条命令遗忘,那只能说明我不够这条命令,它在我使用电脑的过程中并没有那么重要.不要试图掌握所有的命令,就好像上文中提到的`copy`这个命令一样,我实际上使用他的次数不超过20次,我也仅仅只知道他的基础语法而已,包括以上所有的命令,我为什么没写常用参数列表呢?原因很简单,我不记得.

在linux中,我记得`-r`是删除目录,是因为我经常使用,我记得`ls -l`是`ll`的全称,也是因为我经常使用.我在初学cmd的时候,写下了将近1.5w的笔记,那个时候我还是使用word文档写的笔记,估计你看到的时候应该是pdf文档了,或许格式有所瑕疵,还请多多包涵.说远了,那个时候我依然使用word文档来记笔记,所以花费的时间是实打实的,我花费了将近一个星期的时间去学习cmd的命令,并写下了那篇笔记,更不要说我后来还反复丢失过最新版的笔记,只能拿着旧版的备份笔记再做记录(实际上这也是一个美好的经历,这场经历使我有意识的使用云存档和定期备份来保证自己的数据时刻保持安全),可是那将近1.5w的笔记我究竟在日常中使用了多少,我花费的时间,付出的学习成本是否在后来的使用电脑过程中感到值得呢?

所以我姑且将我的体会作为一个劝告来写在这里,不要花费时间去学习你不太可能会用到的东西,我在前面就说过28定律,也许这里是37,或者是46都有可能,但是这都告诉我们不需要学习所有的东西,**我们需要学习我们需要的就好.**

这是我这么久思考后的结果,希望可以帮到你.

5 结束之后

此刻,终于写完了花费了近10天的空闲时间,写下了约4w字.说实话,看到现在我写了这么多东西,我还感到惊奇,咦?原来我不经意间已经写了这么多吗?

我常常在思考,那些地方我是否讲的足够通俗易懂了呢?在不好理解的地方我是否举了好理解的例子,配上了操作的详细图片呢?这么长的目录,这样分级是否合理呢?

因为毕竟这是我手搓的一篇文档,而不是什么官方的权威文档,上面的很多想法都是我个人观点,包括每个部分的最后自学部分更是主观到了极致,更让我觉得废话连篇(bushi).但是考虑考虑还是没有删除,毕竟在我看来确实有一定道理.

关于这篇文章我需要感谢....不对,这篇文章就是我自己写的,我感谢我自己(bushi).感觉是最近看书看的,最近我看到的书的后记部分都喜欢感谢编辑感谢读者的.

说实话再去看一遍我写的东西突然感觉超级羞耻,加上时间问题我也确实没有去捉虫,如果有什么错误的,还请告诉我.我的邮箱是tsdsghbc@qq.com.

啊,如果是图片布局的问题就不要告诉我了,我在写的时候并没有特别注意布局的问题,更何况md文档的初衷就是让笔者尽量少的花费时间在布局的问题上,但是在转化成为pdf的时候难免出现因为图片过大导致大面积空白的问题,如果将来的那一天又想起了这个文档,我大概会来修改一下吧.

就写到这吧.

DSGH

2024年8月9日21点00分