**全网最详细Python虚拟环境pipenv的教程**



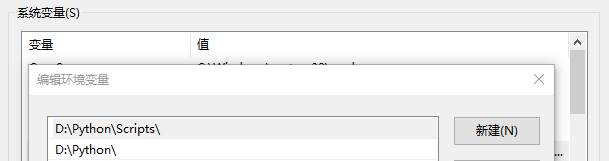
1. **什么是Python虚拟环境**

Python虚拟环境是一个与系统Python环境完全隔离的环境，一个独立的存在。

Python虚拟环境包含了一个特定版本的Python和一些存储相关包的文件夹。我们可以在创建的python虚拟环境中使用pip工具安装任何你需要的模块，该模块和系统Python环境完全不相关。

这里先解释一下什么是系统Python环境。假设在Windows系统中我们在D:\Python目录下安装了Python（只有一个版本的Python）,且设置了相关的环境变量，那么这个Python环境我们就认为是系统Python环境。





1. **为什么需要Python虚拟环境**

这个很简单，这里简单的解释一下。首先我们要知道在一个Python环境中一个版本的Python包只能存在一个，比如在系统环境中存在了Pandas的版本为2.0.2版本，就不能存在Pandas 2.0.0版本的。

假设现在一个A项目需要2.0.2版本的Pandas，B项目需要2.0.0版本的Pandas。我们最好的做法就是A项目使用系统的2.0.2版本的包，B项目使用虚拟环境的2.0.0版本的包，这样我们就不用反复的更新和降低系统中的包版本来达到要求。

这也是为什么在一些开源的项目中，我们看到一个目录下存在requirements.txt的文件。这个文件中列举了固定版本的包，这帮助我们避免了运行项目的不兼容问题，也有助于我们在创建虚拟环境的过程中更准确。

另外使用Python虚拟环境在我们对项目进行打包发布成Exe的时候，不会带入一些不必要的模块，减少打包文件的大小和提升运行速度。

1. **什么是Pipenv**

创建虚拟环境的方法比较多的，常用的模块有Pipenv、Venv、Virtualenv等等。这里我们主要使用Pipenv来创建虚拟环境，也就必不可少的介绍一下这个Pipenv。

首先，Pipenv的官方文档在这：

<https://docs.pipenv.org/>

[Pipenv](https://link.zhihu.com/?target=https://docs.pipenv.org/" \t "https://zhuanlan.zhihu.com/p/_blank)是Kenneth Reitz（也是requests包的作者）在2017年1月发布的Python依赖管理工具，贡献给PyPa之后由PyPa来进行维护。你可以把它看做是pip和virtualenv的组合体，目前Pipenv使用[Pipfile](https://link.zhihu.com/?target=https://github.com/pypa/pipfile" \t "https://zhuanlan.zhihu.com/p/_blank)依赖管理文件来替代过于简陋的 requirements.txt 文件。也就是说Pipfile中列出了项目运行所需的所有软件包和它们的依赖性，这个文件作为环境的快照。

1. **Pipenv的安装**

我们在系统环境中安装Pipenv模块，命令如下（系统Python已经加入Windows的环境变量）：

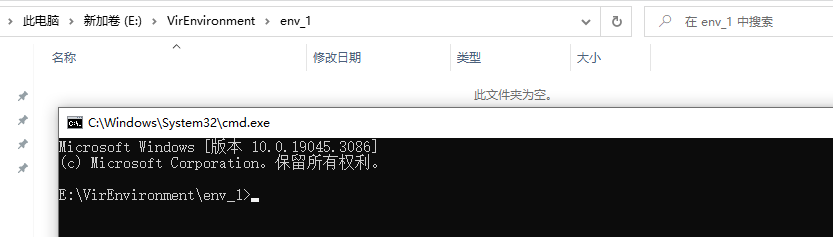


安装成功即可。

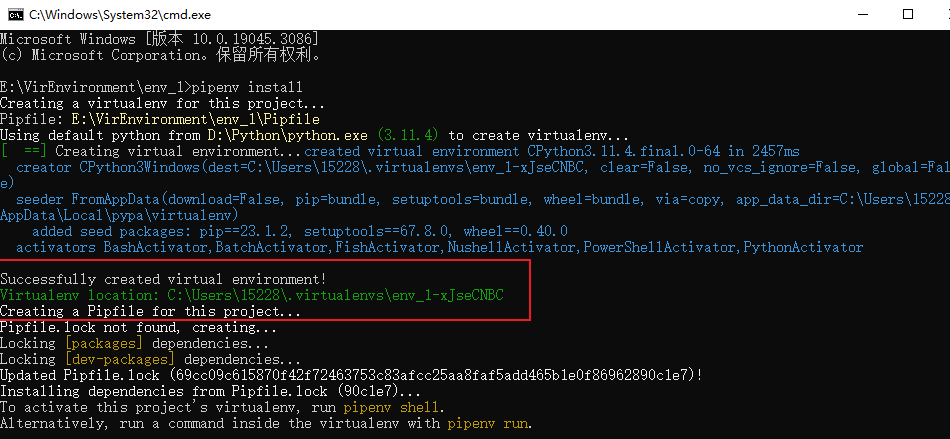
1. **使用Pipenv创建虚拟环境**

接下来就是使用Pipenv来创建虚拟环境，因为要保存Pipfile文件，因此我们得先创建一个目录：E:\VirEnvironment\env\_1

接着在此目录中打开cmd窗口：



输入命令：pipenv install



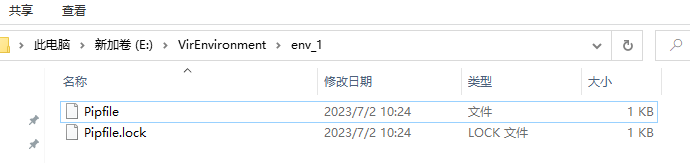
截图中标注的部分可以看出，我们创建的虚拟环境的路径为：

C:\Users\15228\.virtualenvs\env\_1-xJseCNBC

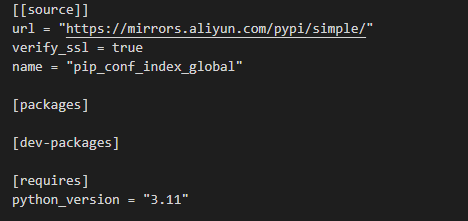
该目录下我们可以看到我们得虚拟环境，虚拟环境的目录env\_1-xJseCNBC，是由项目所在文件夹env\_1加上一个随机的字符串创建的。



回过头来看一下我们的项目目录情况，可以看出env\_1下生成了一些Pipfile文件。



Pipfile中存储着一些信息：



[source]: Python包下载路径，这里设置的是国内源;

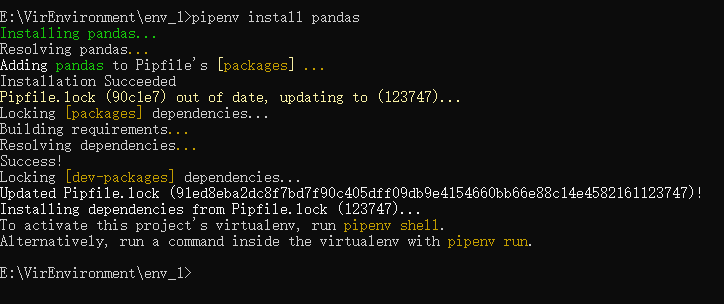
[packages]:罗列相应的包名称;

[dev-packages]: 罗列相应的开发包名称;

[requires]: Python版本号。

创建虚拟环境之后，假设我们要安装包，可以紧接着输入命令：

pipenv install pandas

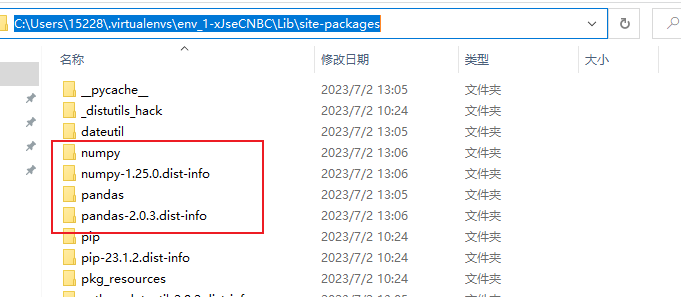


可以看出，我们正确安装了Pandas，此时我们打开Pipfile.lock文件内容：

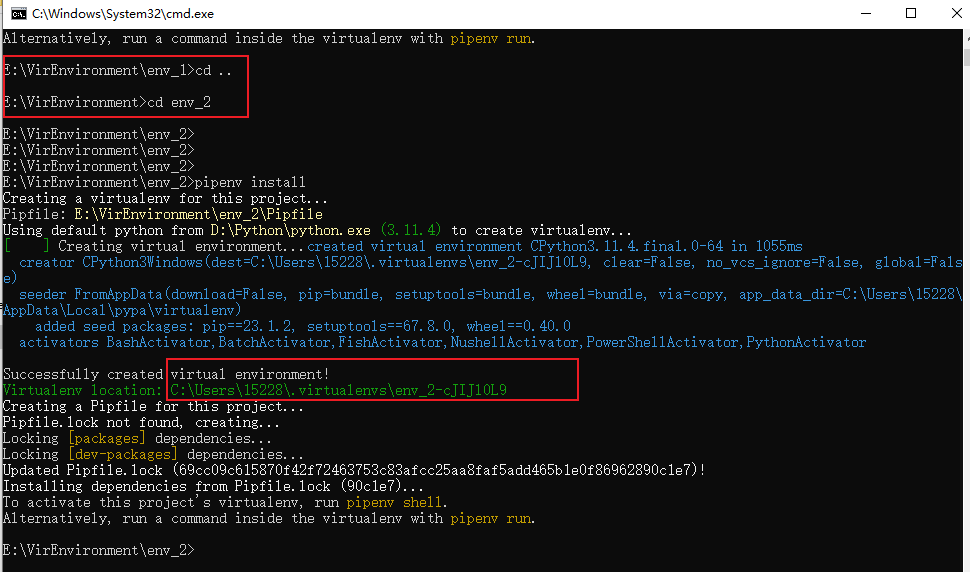


记录的是一些包名的内容、版本号以及hash值，用于恢复使用。也就是说从Pipfile文件添加或删除安装的包，会生成Pipfile.lock来锁定安装包的版本和依赖信息，通过pipfile.lock文件，可以精确恢复以来的版本。

对应的虚拟环境目录下已经有Pandas包，以及Pandas的一些依赖包：

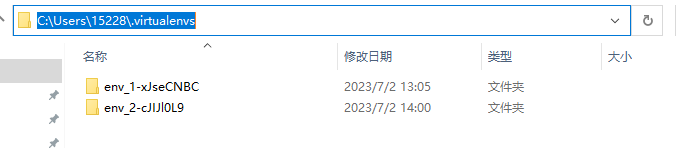


这样我们就完成了虚拟环境的安装，假设我们还要新建一个新的虚拟环境，只需要新建一个新的文件夹如env\_2,然后进入这个文件夹，执行pipenv install命令即可：



1. **更改虚拟环境创建目录**

细心的同学可能已经发现，我们每一次新建的目录都会在C盘的目录中，比如之前创建的环境：



如果C盘储存紧张的时候，这个方法就不太友好了，我们可以使用以下方法进行更改目录，假设我们现在要将虚拟环境安装此目录：

E:\Works\DevProject

我们得步骤如下：

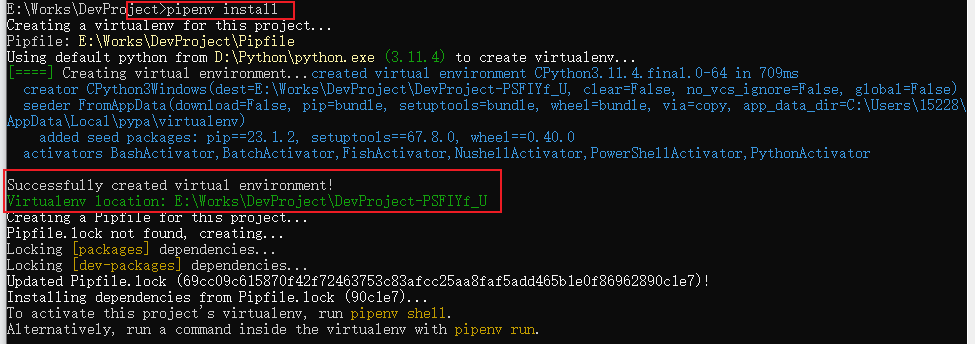
1. cd 到目录中E:\Works\DevProject



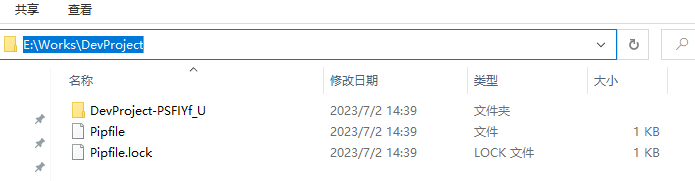
1. 执行set WORKON\_HOME=E:\Works\DevProject



1. 执行pipenv install



可以看出我们得虚拟环境已经更换了位置：



**注意：设置** WORKON\_HOME的时候只对本次虚拟环境有效果，也就是说再次创建新的虚拟环境的时候，还得设置 WORKON\_HOME指明位置。

1. **虚拟环境的使用**

创建好虚拟环境之后，最重要的是如何使用。首先是一些常用的命令：

(1) pipenv --venv :显示虚拟环境路径

(2) pipenv --where: 显示本地工程路径

(3) pipenv --py: 显示Python解释器的路径信息

(4) pipenv install 创建虚拟环境

(5) pipenv install pacgage-name:安装包

(6) pipenv uninstall pacgage-name 删除包

(7) pipenv uninstall --all 卸载所有包

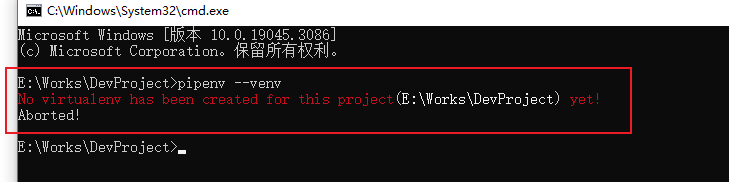
(8) pipenv graph 查看包依赖

(9) pipenv lock 生成lockfile

(10) pipenv --rm 删除虚拟环境

(11) pipenv run python xxx.py 虚拟环境运行python

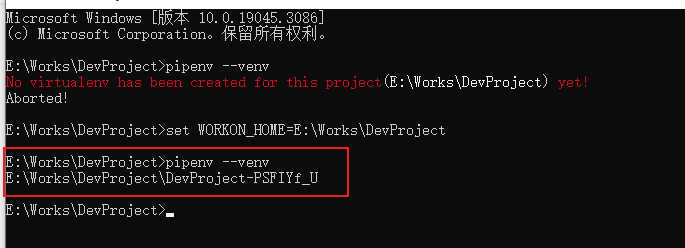
我们来试一下第一个命令，进入到该目录，运行命令：



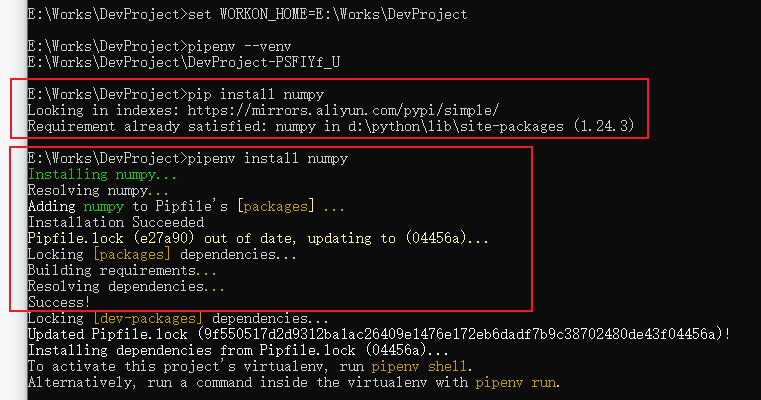
发现报错了，说我们没有为这个工程创建虚拟环境，可是我们明明创建了呀。原来是我们在创建虚拟环境的时候设置了：

WORKON\_HOME=E:\Works\DevProject

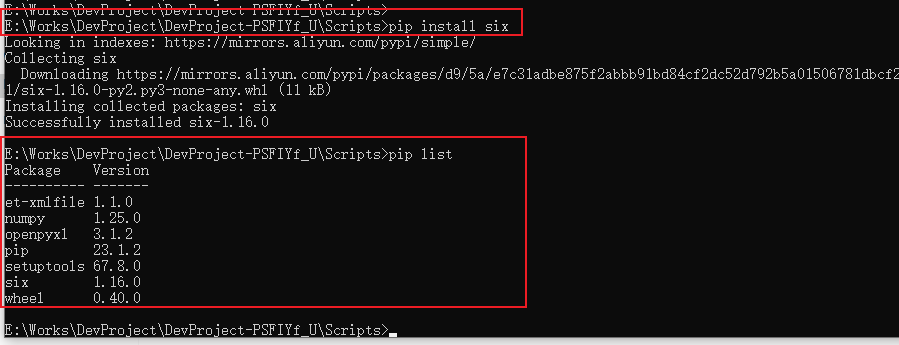
因此我们在运行的时候，也要进行这样的操作：



这样就没有报错了，之后我们就可以安装我们需要的Python模块了，比如安装我们常用的numpy:



一开始使用的是pip来安装模块的，显示已经安装，这是因为pip命令调用的是系统环境中的pip命令的，对于刚创建的虚拟环境需要使用pipenv命名，如果实在不想使用pipenv命令，我们可以进入到虚拟环境pip.exe所在的目录安装模块，如下所示：

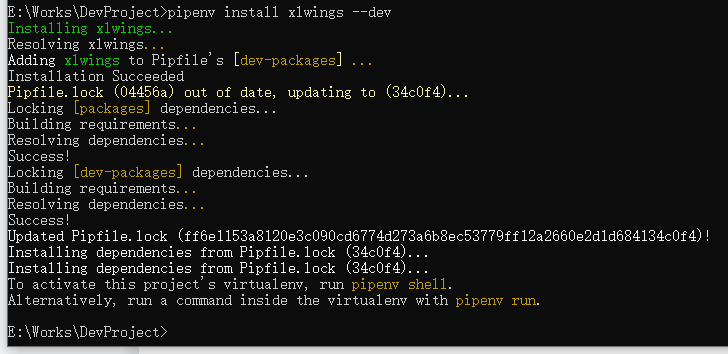


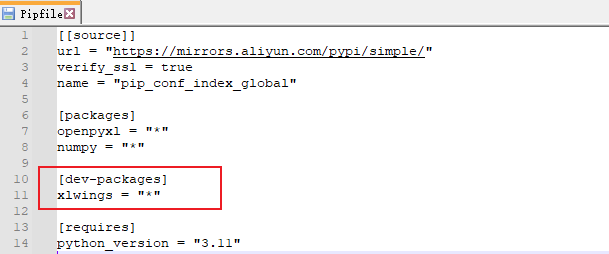
**使用pip来安装模块的时候，Pipfile和Pipfile.lock文件是不会进行同步更新的，因此需要再次使用pipenv install package-name才能更新。**

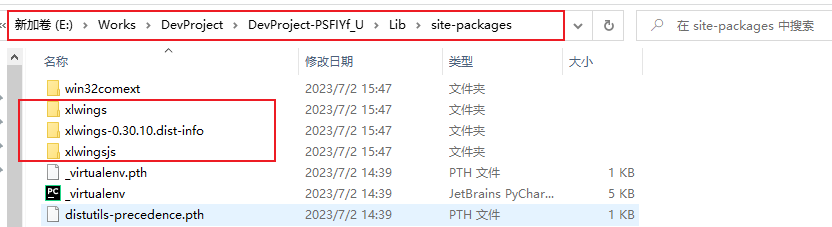
我们再来看一个命令：

pipenv install xlwings --dev

这个命令的意思是安装xlings这个包，且将xlwings这个包的名字记录到Pipfile的[dev-packages]下，说明这个包是开发环境独有的，上生产的时候可能是不需要这个包的，仅仅作为记录使用，实际上这个xlwings这个包还是安装到虚拟环境里面的，截图如下：







1. **生成requirements.txt文件**

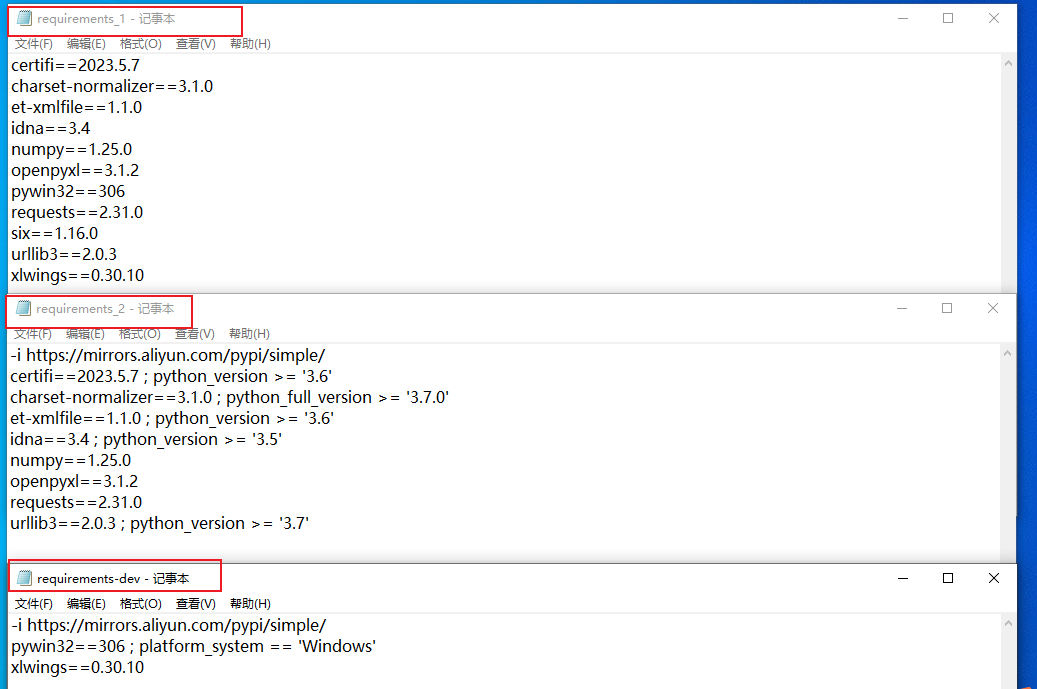
我们经常需要将所需项依赖包的版本进行打包，达到快速发布的目的，这样别人在使用的时候就可以快速创建环境，这里有两种方法：

方法1：pipenv run pip freeze > requirements\_1.txt  
方法2：pipenv requirements > requirements\_2.txt

方法3：pipenv requirements --dev-only > requirements-dev.txt

我们运行一下这三个命令，看一下生成的文件有什么不同之处：

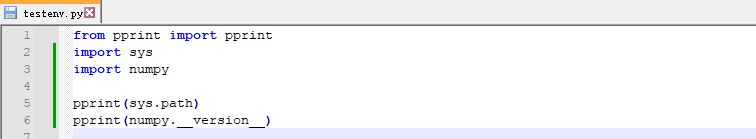




可以看出第一种方法罗列了虚拟环境中的所有的包，第二种方法罗列了虚拟环境中非开发环境的包且包含了Python第的版本号，第三种方法只罗列的是开发包。

1. **使用虚拟环境运行脚本**

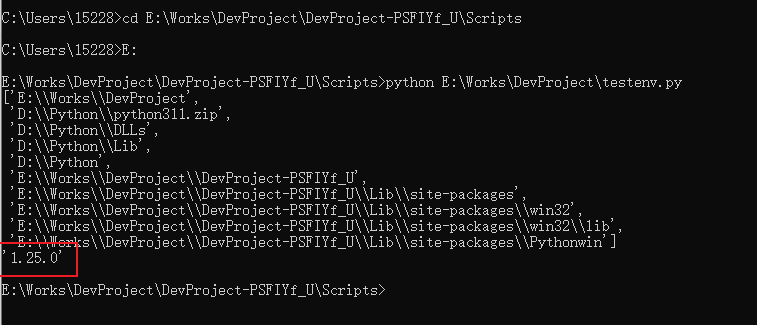
假设现在有个文件，我们怎么使用虚拟环境中的Python解释器来运行它呢？文件内容如下：



有几种运行方式：

**方案一：**

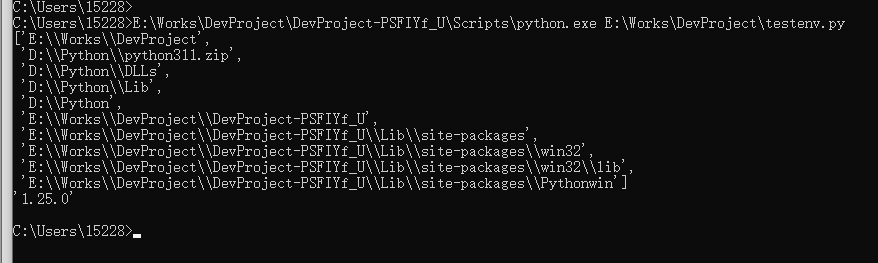
1. cd 到 E:\Works\DevProject\DevProject-PSFIYf\_U\Scripts
2. 运行python.exe E:\Works\DevProject\testenv.py



可以看出，我们这里打印的numpy的版本为1.25.0（系统Python的numpy的版本为1.24.0）。这就说明，我们使用的Python解释器是虚拟环境的。

**方案二：**

直接运行脚本,效果是一样的：



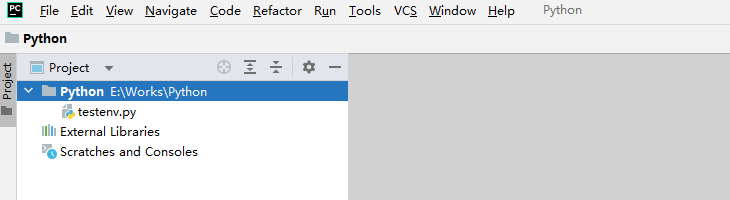
1. **Pycharm中设置虚拟环境**

Pycharm中设置虚拟环境是比较简单的，我们先将测试文件移动一个位置到E:\Works\Python

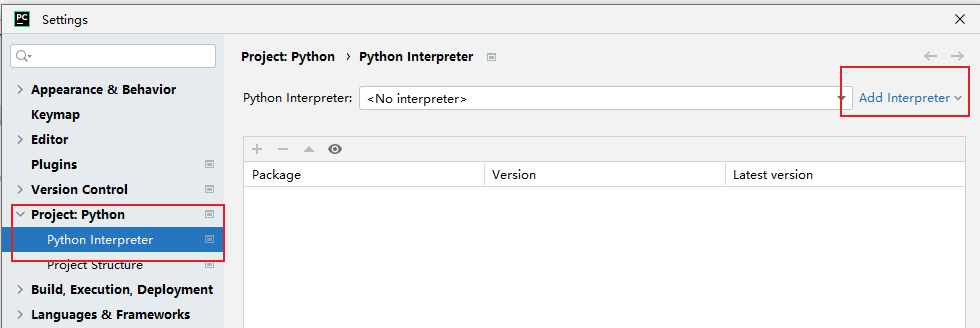
下：



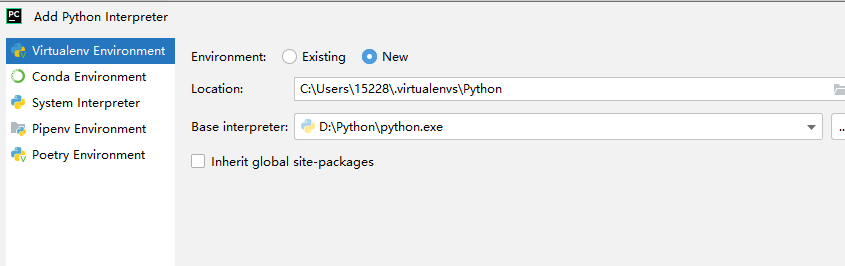
1. 打开Pycharm，设置项目文件目录：



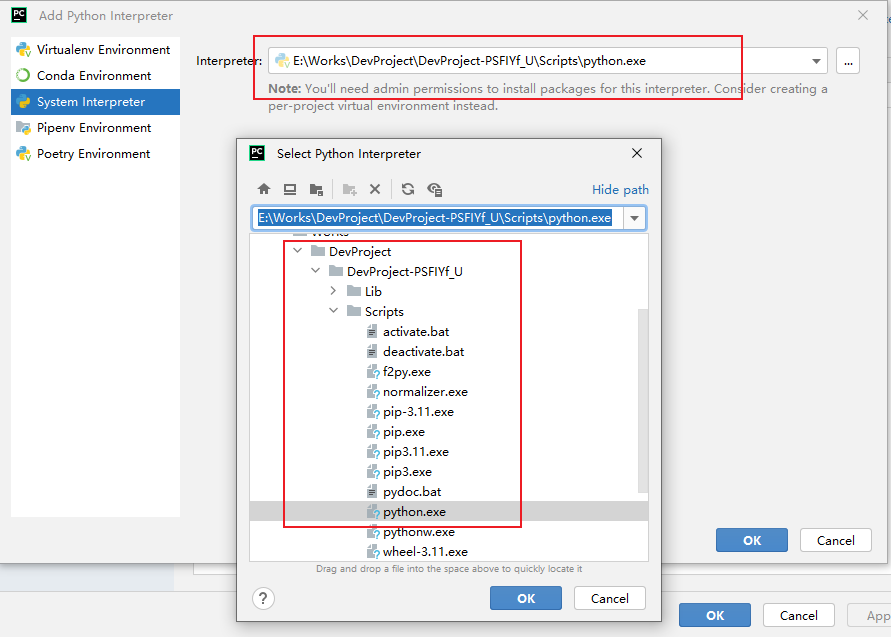
1. 依次点击File->Settings->Project:Python->Python Interpreter:



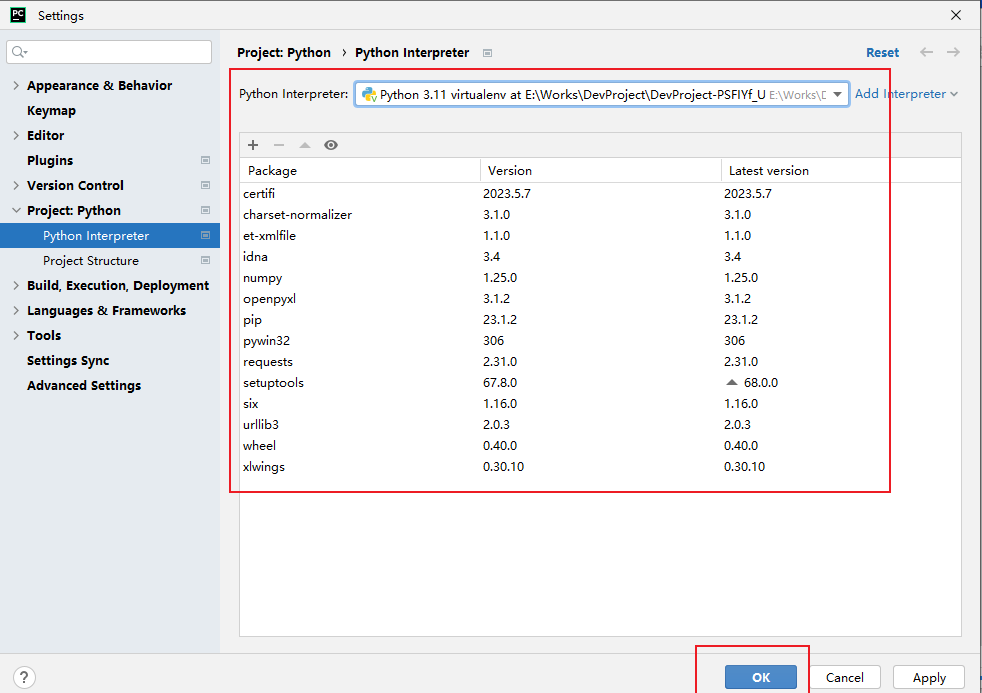
1. 依次点击add Interpreter-> add local Interpreter,弹出如下界面



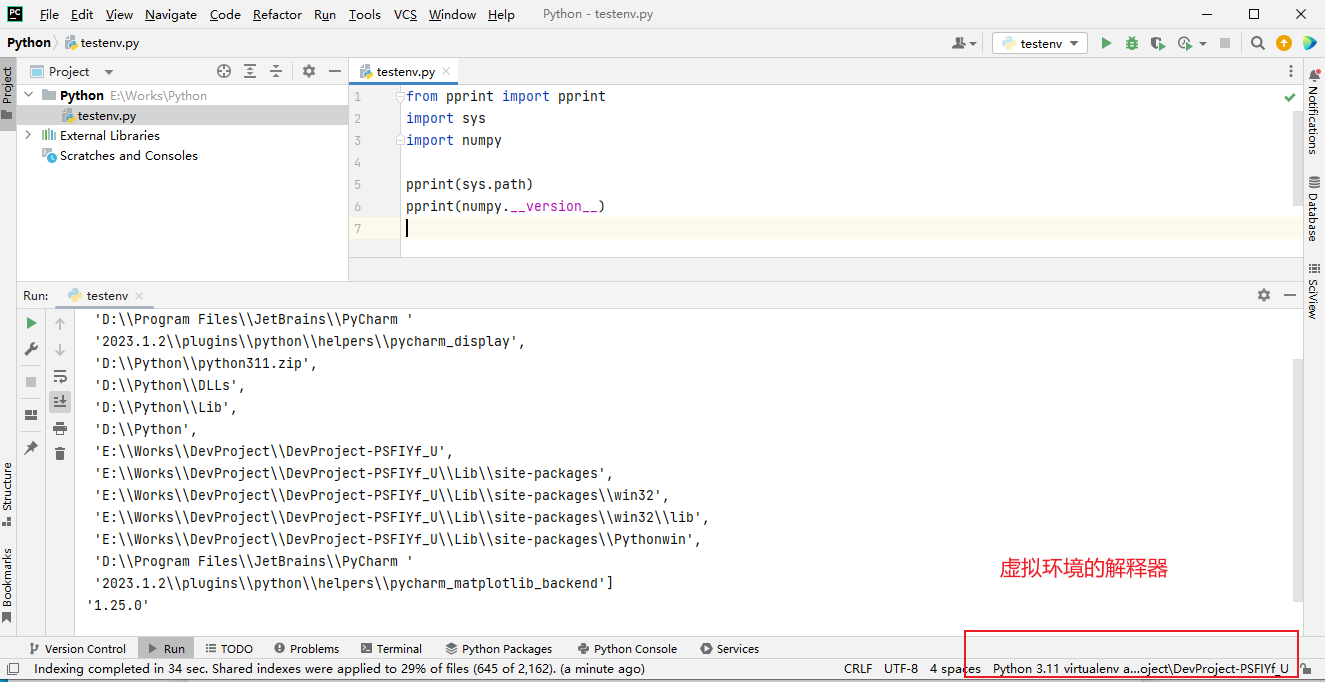
1. 选择System Environment按照下图设置，最后点击确定：



可以看到Python包的一些情况，最后点击Ok即可：



运行脚本如下所示：



完成使用虚拟环境中的Python来执行代码,完美。

1. **总结**

本次手把手教学了基于Pipenv创建虚拟环境，使用虚拟环境的操作，相信大家已经掌握了其中的精髓，如果要掌握上述知识建议跟着教程进行实战，有什么疑问大家一起讨论交流，下次再见。

1. **参考文献**

https://docs.pipenv.org/