𝐇𝐨𝐰 𝐭𝐨 𝐒𝐢𝐦𝐩𝐥𝐢𝐟𝐲 𝐏𝐲𝐭𝐡𝐨𝐧 𝐈𝐦𝐩𝐨𝐫𝐭𝐬 𝐰𝐢𝐭𝐡 𝐄𝐱𝐩𝐥𝐢𝐜𝐢𝐭 𝐏𝐚𝐜𝐤𝐚𝐠𝐢𝐧𝐠?  
  
When developing Python projects, it is always recommended to package them.  
  
Simply put, if a project is packaged, you can import stuff from it.  
  
While 𝐏𝐲𝐭𝐡𝐨𝐧 3.3+ provides Implicit Namespace Packages — a directory with modules is considered a package by default, it is still advised to create an explicit \_\_𝐢𝐧𝐢𝐭\_\_.𝐩𝐲 file.  
  
A couple of major benefits of doing this are that it helps in:  
- Explicitly specifying which classes/functions can be imported from the package.  
- Avoiding redundant imports.  
  
For instance, consider the directory structure in 𝐝𝐢𝐫𝐞𝐜𝐭𝐨𝐫𝐲\_𝐨𝐥𝐝 in the image below:  
- 𝐭𝐫𝐚𝐢𝐧.𝐩𝐲 has a 𝐓𝐫𝐚𝐢𝐧𝐢𝐧𝐠 class.  
- 𝐭𝐞𝐬𝐭.𝐩𝐲 has a 𝐓𝐞𝐬𝐭𝐢𝐧𝐠 class.  
  
We can directly import the 𝐓𝐫𝐚𝐢𝐧𝐢𝐧𝐠 and 𝐓𝐞𝐬𝐭𝐢𝐧𝐠 class in 𝐩𝐢𝐩𝐞𝐥𝐢𝐧𝐞.𝐩𝐲 as follows:  
- 𝐟𝐫𝐨𝐦 𝐦𝐨𝐝𝐞𝐥.𝐭𝐫𝐚𝐢𝐧 𝐢𝐦𝐩𝐨𝐫𝐭 𝐓𝐫𝐚𝐢𝐧𝐢𝐧𝐠  
- 𝐟𝐫𝐨𝐦 𝐦𝐨𝐝𝐞𝐥.𝐭𝐞𝐬𝐭 𝐢𝐦𝐩𝐨𝐫𝐭 𝐓𝐞𝐬𝐭𝐢𝐧𝐠  
  
While this will work as expected, the problem is that we have to explicitly import the specific class from each of the modules.  
  
But using explicitly packaging by defining the \_\_𝐢𝐧𝐢𝐭\_\_.𝐩𝐲 file can simplify this.  
  
As shown in 𝐝𝐢𝐫𝐞𝐜𝐭𝐨𝐫𝐲\_𝐧𝐞𝐰:  
- We first explicitly package the directory by creating an \_\_𝐢𝐧𝐢𝐭\_\_.𝐩𝐲 file.  
- Next, we specify the imports directly in this file.  
  
Now, instead of writing redundant imports, we can directly import the intended classes from the 𝐦𝐨𝐝𝐞𝐥 package.  
  
This simplifies your imports.  
  
Also, as discussed earlier, an \_\_𝐢𝐧𝐢𝐭\_\_.𝐩𝐲 file lets you explicitly specify which classes/functions can be imported from the package, which, otherwise, will not be evident.  
  
This simplifies things for other users of the project.  
  
Want to learn more about Packaging? Ask any question you have and get AI-powered explanations and 💡 recommendations for the best learning resources. 👉 <https://lnkd.in/gJwd3ZVa>

A screenshot of a program

Description automatically generated