Using "from module import something"

There are many times when you just want to import part of a module or library. Let's see how Python accomplishes this:

```
from functools import lru_cache
```

What the code above does is allow you to call lru_cache directly. If you had imported just functools the normal way, then you would have to call lru_cache using something like this:

```
functools.lru_cache(*args)
```

Depending on what you're doing, the above might actually be a good thing. In complex code bases, it's quite nice to know where something has been imported from. However, if your code is well maintained and modularized properly, importing just a portion from the module can be quite handy and more succinct.

Of course you can also use the from method to import everything, like so:

```
from os import *
```

This is handy in rare circumstances, but it can also really mess up your namespace. The problem is that you might define your own function or a top level variable that has the same name as one of the items you imported and if you try to use the one from the os module, it will use yours instead. So you end up with a rather confusing logic error. The Tkinter module is really the only one in the standard library that I've seen recommended to be imported in total.

If you happen to write your own module or package, some people recommend

importing everything in your __init__.py to make your module or package

easier to use. Personally I prefer explicit to implicit, but to each their own.

You can also meet somewhere in the middle by importing multiple items from a package:

```
from os import path, walk, unlink
from os import uname, remove
```

In the code above, we import five functions from the os module. You will also note that we can do so by importing from the same module multiple times. If you would rather, you can also use parentheses to import lots of items:

```
from os import (path, walk, unlink, uname,
            remove, rename)
```

This is useful technique, but you can do it another way too:

```
from os import path, walk, unlink, uname, \
            remove, rename
```

The backslash you see above is Python's line continuation character, which tells Python that this line of code continues on the following line.