# WHAT ARE PACKAGES?

## Packages are Modules

Packages are modules (but modules are not necessarily packages)

They can contain

- modules
- packages (called sub-packages)

If a module is a package, it must have a value set for `__path__`

After you have imported a module, you can easily see if that module is a package by inspecting the `__path__` attribute (empty → module, non-empty → package)

## Packages and File Systems

Remember that modules do not have to be entities in a file system (loaders, finders)

By the same token, packages do not have to be entities in the file system

Typically they are - just as typically modules are file system entities

But packages represent a hierarchy of modules / packages

```
pack1.mod1
pack1.pack1_1.mod1_1
```

dotted notation indicates the path hierarchy of modules / packages

and is usually found in `__path__`

## Importing Nested Packages

If you have a statement in your top-level program such as:

```
import pack1.pack1_1.module1
```

The import system will perform these steps:

- imports `pack1`
- imports `pack1.pack1_1`
- imports `pack1.pack1_1.module1`

The `sys.modules` cache will contain entries for:

```
pack1
pack1.pack1_1
pack1.pack1_1.module1
```

The namespace where the import was run contains:

```
pack1
```

## File System Based Packages

Although modules and packages can be far more generic than file system based entities, it gets complicated!

If you're interested in this, then the first document you should read is PEP302

In this course we're going to stick to traditional file based modules and packages

## File Based Packages

→ package paths are created by using file system directories and files

Remember: a package is simply a module that can contain other modules/packages

On a file system we therefore have to use directories for packages

The directory name becomes the package name

So where does the code go for the package (since it is a module)?

`__init__.py`

# `__init__.py`

To define a package in our file system, we must:

create a directory whose name will be the package name

create a file called `__init__.py` inside that directory

That `__init__.py` file is what tells Python that the directory is a package as opposed to a standard directory

(if we don't have an `__init__.py` file, then Python creates an implicit namespace package)

– we'll discuss that later

## What happens when a file based package is imported?

```
app/
    pack1/
        __init__.py
        module1.py
        module2.py
```

```
import pack1
```

the code for `pack1` is in `__init__.py`

that code is loaded, executed and cached in `sys.modules` with a key of `pack1`

it's just a module!

the symbol `pack1` is added to our namespace referencing the same object

it's just a module!

but, it has a `__path__` property → file system directory path (absolute)

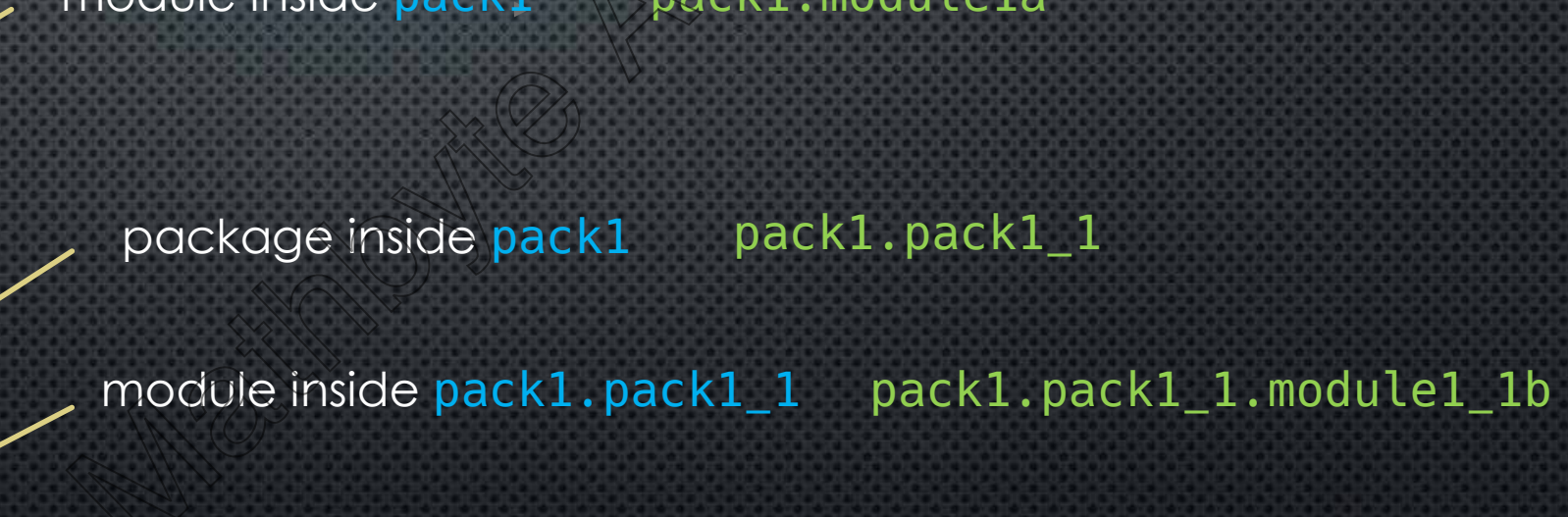
also has a `__file__` property → file system path to `__init__.py` (absolute)

## Nested Packages

Packages can contain modules as well as packages

```
app/
    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```



# `__file__`, `__path__` and `__package__` Properties

Modules have `__file__` and `__package__` properties

`__file__` is the location of module code in the file system

`__package__` is the package the module code is located in

(an empty string if the module is located in the application root)

If the module is also a package, then it also has a `__path__` property

`__path__` is the location of the package (directory) in the file system

```
app/
    module.py

    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```

```
module.__file__ → …/app/module.py
module.__path__ → not set
module.__package__ → ''

pack1.__file__ → …/app/pack1/__init__.py
pack1.__path__ → …/app/pack1
pack1.__package__ → pack1

pack1.module1a.__file__ → …/app/pack1/module1a.py
pack1.module1a.__path__ → not set
pack1.module1a.__package__ → pack1
```

```
app/
    module.py

    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```

```
pack1.pack1_1.__file__ → …/app/pack1/pack1_1/__init__.py
pack1.pack1_1.__path__ → …/app/pack1/pack1_1
pack1.pack1_1.__package__ → pack1.pack1_1

pack1.pack1_1.module1_1a.__file__ → …/app/pack1/pack1_1/module1_1a.py
pack1.pack1_1.module1_1a.__path__ → not set
pack1.pack1_1.module1_1a.__package__ → pack1.pack1_1
```

## What gets loaded during the import phase?

```
app/
    module.py

    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```

```
import pack1.pack1_1.module1_1a
```

at the very least:

`pack1` is imported and added to `sys.modules`

`pack1_1` is imported and added to `sys.modules`

`module1_1a` is imported and added to `sys.modules`

but, modules can import other modules!

`pack1.__init__.py` could import other modules/packages

`pack1_1.__init__.py` could import other modules/packages

`module1_1a.__init__.py` could import other modules/packages

For example...

```
app/
    module.py

    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```

```
# pack1.__init__.py

import pack1.module1a
import pack1.module1b
```

```
import pack1.pack1_1.module1_1a
```

Just as before:

`pack1` is imported and added to `sys.modules`

`pack1_1` is imported and added to `sys.modules`

`module1_1a` is imported and added to `sys.modules`

but now also:

`module1a` is imported and added to `sys.modules`

`module1b` is imported and added to `sys.modules`

For example...

```
app/
    module.py

    pack1/
        __init__.py
        module1a.py
        module1b.py

        pack1_1/
            __init__.py
            module1_1a.py
            module1_1b.py
```

```
# pack1.__init__.py

import pack1.module1a
import pack1.module1b
```

```
import pack1
```

`pack1` is imported and added to `sys.modules`

`module1a` is imported and added to `sys.modules`

`module1b` is imported and added to `sys.modules`

# Code