WHY PACKAGES?

Code Organization, Ease of Use…

Suppose you have 50 different functions and classes in your program

api.py (single file)

```
connect                 User                audit_endpoint
execute_no_result       UserProfile
execute_single_row      Users               Logger
execute_multi_row
                        BlogPost            validate_email
normalize_string        BlogPosts           validate_phone
convert_str_to_bool                         validate_name
format_iso_date         RouteTable
current_time_utc        Configuration       etc…

authenticate            JSONEncoder
validate_token
get_permissions         UnitTests
authorize_endpoint
```

in **one** file???

Start with Modules…

```
api/
    api.py
    dbutilities.py
    jsonutilities.py
    typeconversions.py
    validations.py
    authentication.py
    authorization.py
    users.py
    blogposts.py
    logging.py
    unittests.py
```

better…

but still unwieldy – everything is at the top level

too many imports:

```
import dbutilities
import jsonutilities
import typeconversions
import validations
import authentication
import authorization
import users
```

etc…

certain modules could be broken down further:

dbutilities → connections, queries

users → User, Users, UserProfile

certain modules belong "together":

authentication, authorization → security

So, Packages...

```
api/
    api.py
    dbutilities.py
    jsonutilities.py
    typeconversions.py
    validations.py
    authentication.py
    authorization.py
    users.py
    blogposts.py
    logging.py
    unittests.py
```

```
api/
    utilities/
        __init__.py
        database/
            __init__.py
            connections.py
            queries.py
        json/
            __init__.py
            encoders.py
            decoders.py
    security/
        __init__.py
        authentication.py
        authorization.py
    models/
        __init__.py
        users/
            __init__.py
            user.py
            userprofile.py
```

Another Use Case

You have a module that implements 2 functions/classes for users of the module

Those two objects require 20 different helper functions and 2 additional helper classes

From module developer's perspective:

much easier to break the code down into multiple modules

From module user's perspective:

they just want a single import for the function and the class

i.e. it should look like a single module

Module Developer's Perspective

```
mylib/
    __init__.py
    submod1.py
    submod2.py
    subpack1
        __init__.py
        pack1mod1.py
        pack1mod2.py
```

function to be exported to user lives here

class to be exported to user lives here

Smaller code modules, with a specific purpose, are easier to write, debug, test, and understand

Module User's Perspective

```
mylib/
    __init__.py
    submod1.py
    submod2.py
    subpack1
        __init__.py
        pack1mod1.py
        pack1mod2.py
```

function to be exported to user lives here (submod1.py)

class to be exported to user lives here (pack1mod2.py)

User should not have to write:

```
from mylib.submod1 import my_func
from mylib.subpack1.pack1mod2 import MyClass
```

Much easier for user if they could write:

```
from mylib import my_func, MyClass
```

or, simply

```
import mylib
mylib.my_func()   mylib.MyClass()
```

Using `__init__.py`

We can use packages' `__init__.py` code to export (expose) just what's needed by our users

Example:

```
mylib/
    __init__.py
    submod1.py
    submod2.py
    subpack1
        __init__.py
        pack1mod1.py
        pack1mod2.py
```

function to be exported to user lives here (submod1.py)

class to be exported to user lives here (pack1mod2.py)

```python
# mylib.__init__.py

from mylib.submod1 import my_func
from mylib.subpack1.pack1mod2 import MyClass
```

User uses it this way:

```python
import mylib

mylib.my_func()
mylib.MyClass()
```

our internal implementation is "hidden"

We'll cover this in the next video

So, why Packages?

ability to break code up into smaller chunks, makes our code:

- easier to write
- easier to test and debug
- easier to read/understand
- easier to document

just like books are broken down into chapters, sections, paragraphs, etc.

but they can still be "stitched" together

- hides inner implementation from users
- makes their code
  - easier to write
  - easier to test and debug
  - easier to read/understand