book→
User → from Users

Modules

⇒ Real Project are so big:-

→ We Can't Keep all Code in simple file

→ ~~We Can't Keep Project in a simple fil~~ We Break it in a smaller tasks.

Modular Programming

↳ simplicity, maintainability, Reuseability

⟹ Python way:

→ The Smallest task in Python is called a Module

Single .py file focusing on specific Task.

→ Package → Group of modules (.py files). Bigger Subproblem scope

→ Scoping → each module has different namespace → No name Collision.

book→
User → from users

Modules

=> Real Project are so big:

→ We can't keep all code in simple file

→ ~~We Can't Keep Project in a simple fi~~ We Break it in a smaller tasks.

Modular Programming

↳ simplicity, maintainability, Reuseability

=> Python way:

→ The Smallest task in Python is called a Module

Single .py file focusing on specifing Task.

→ Package → Group of modules (py files) - Bigges Subproblem scope

→ Scoping → each module has different namespace → No name Collision.

Hospital System

Sub-tasks

↳ We wrote all in Single file → This wrong

- Utilities.py
  - ↳ input_valid_int (msg, start=0, end = None)
- Patient.py
- → hospitalmgr.py
- → frontendmgr.py

→ make it more easier to give everyone a Task.

Scanned with CamScanner

Modules : Develop , Use and install:

↳ Develop your own module ⟶ (Like the hospital system)

2) Make use of built-in modules

↳ We already use builtin modules ⟶ max, len, dir
has more built in modules that are focused on specific Task.

3) we can install external Packages.

We Want to Use Math module :-

```
Import math
→ Print (math.sqrt(16)) ⟶ get Square.
→ Print (math. factorial (5)) ⟶ get Factorial of a Number.
→ Print (math. Pi) ⟶ Return 3.14
→ Print ( math. cos (math.Pi/2)) ⟶ we use value as Pi Not 90 or 180.
```

We can Rename a module and Rename it.

```
↳ import math as xxx
↳ Print ( xxx . Pi)
```

⟶ use very short Name for more easier way or To prevent distraction when there is multiple Classes with the Same name.

We can Import functions or Import variables and been Seen in my script :-

```
↳ From math Import PI, factorial
Print (PI)
Print (factorial (5))
```

⟶ We should be aware of Names to avoid name Collisions.

⇒ don't use ~~this~~ from math import * ⟶ means Import all :-

→ inside a function we can import a module to use it inside this Scope only. To avoid use it in another function or main.

OS and Sys modulas

⇒ **Enviroment Variable :-**

↳ Name ⟶ Value that a Process may Acess to get some info (For Configuration)

In Python → echo $PWD ⇒ Return current working Dir.

⇒ **Python Path :-** is an Enviroment Variable, its Value are List of directories

↳ it's Used to ADD Path For User Defined modules ⟶ Primary Reason

it's Dir added to sys. Path directory list.

→ Note : through os module, we can access enviroment variable From Python.

OS module

→ Import os

has every thing Relative to enviroment Values

→ Print(List(os.environ.keys())

↳ Return values as ['Path', 'Home', 'User', 'PWD'...]

→ Print(os environ['Home'] → it will Return an error If it's Not Founded

↳ So we use $\boxed{\text{get}}$ Better

→ When you make changes it effect only your Local Session. → os.environ doesn't overwrite the System Vars.

→ os.environ.get('Path') → Return executable Files only

most Imp.

→ os.environ.get('Python Path') → It has directories ⟶ we use For speifing where are other modules.

[redacted] Inport sys → Parameters specific to system

#Search Path for modules   # Script dict (or Current for Interactive)

#initialize from the enviroment Variable PythonPath

Print(sys.Path)

Creating Modules

Module → Python file

↳ We can make it few steps → Create afile with some functions

↓ "Import this file to your file to work with it"

Print("fileName".__file__") → Return it's Path.

→ When you import one function from Library

↳ you cant use __file__ or any other functions in it

Module Path Search

⇒ The Module Search Path :

↳ When The interpreter executes import ourlib → ex

↳ Search In 3 Places

First Location: Script directory → Pycharm /latest/ File1.Py

* Observe not the Current directory

Second Location: List of directories contained in PythonPath.

Third Location: Installation - dependent list of directory

↳ All of them are in sys.Path in order.

Note: if the files are not in the same directory you can't import them

↳ It will Return "Module Not founded error"

⇒ Because it's Not founded In PythonPath or installation dirs

→ We Can Import a module By adding it's Path

- Import sys
- sys.Path.append ("Path of the file")

↳ list of directories

→ Note: Be ~~Cause~~ Care Full You Name module as a Pre-Named Modules / or Founded one's

Note In windows we use (//) → in my windows.

Difference Between Module and script

From Python Prespective :-

Modules : are intended to be imported as a Library

scripts : are Top level files acting like an application.

Note: When we import any file it Run all the Content it have first then Run The file it's imported in.

Scanned with CamScanner