CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

- **CONTENTS**
  - Outcomes
- What is a module?
  - Using a Module
    - Importing a module

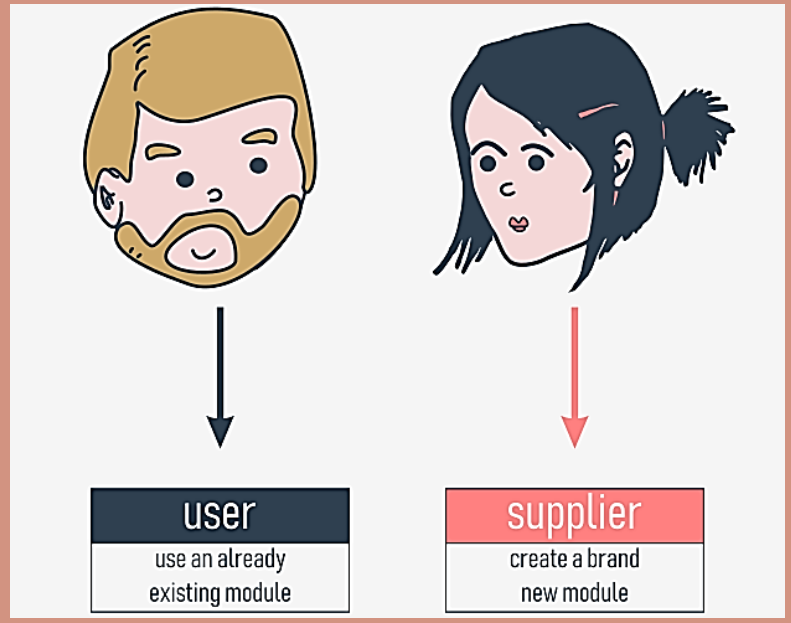
- **CONTENTS**
  - Outcomes
- What is a module?
  - Using a Module
    - Importing a module

- **Python packaging ecosystem and how to use it**
- **The PyPI repo: the Cheese Shop**
- **How to install *pip***
- ***pip* on MS Windows**
- ***pip* on Linux**
- **Dependencies**
- **How to use *pip***

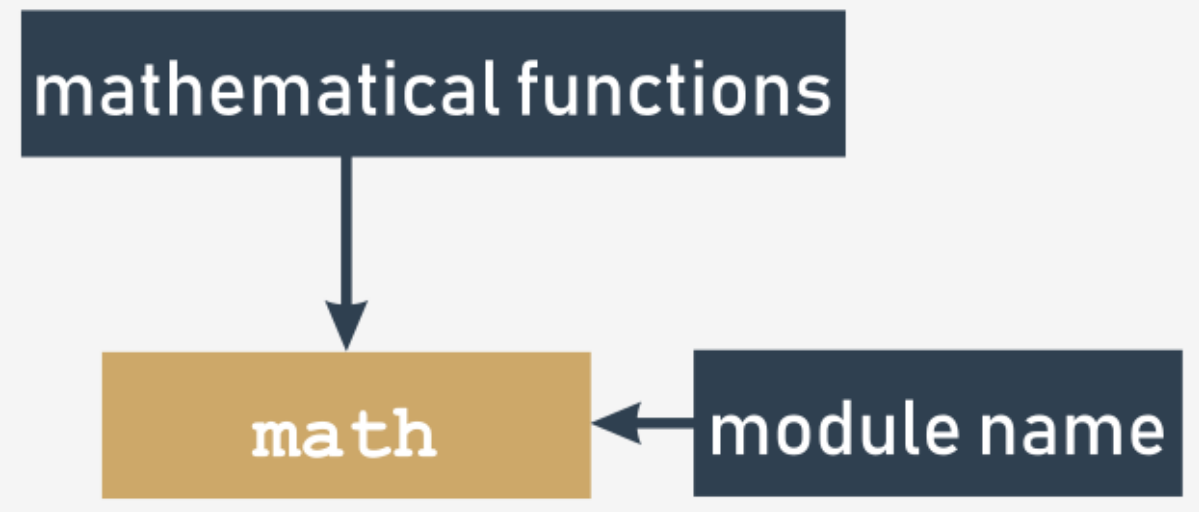
- In this module, you will learn about:
  - importing and using Python modules;
  - using some of the most useful Python standard library modules;
  - constructing and using Python packages;
  - PIP (Python Installation Package) and how to use it to install and uninstall ready-to-use packages from PyPI.

- In Python, Modules are simply **files with the ". py" extension containing Python code that can be imported inside another Python Program**. In simple terms, we can consider a module to be the same as a code library or a file that contains a set of functions that you want to include in your application.

- A module is **a file containing Python definitions and statements**, which can be later imported and used when necessary.
- A module is identified by its name.
- A large number of modules is delivered together with Python itself.

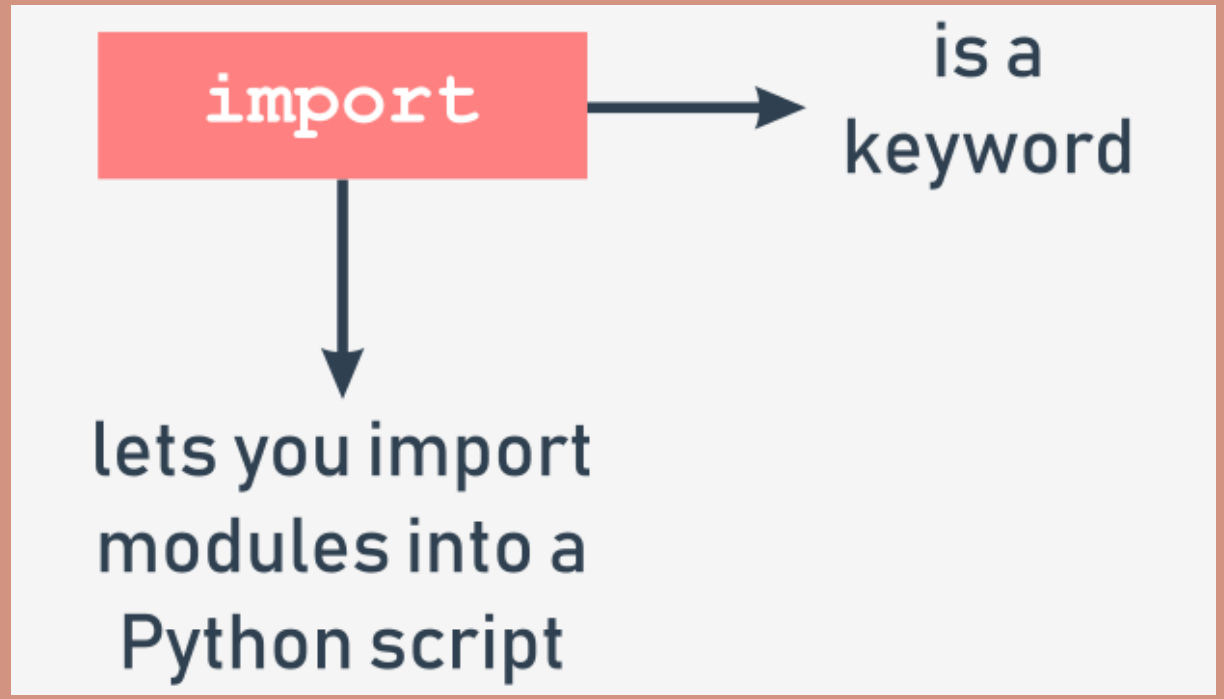


- Each module consists of entities (like a book consists of chapters).
- These entities can be functions, variables, constants, classes, and objects. If you know how to access a particular module, you can make use of any of the entities it stores.
-



- There is a frequently used module, named math. The module contains a rich collection of mathematical functions, like sin() or log().

- To make a module usable, you must **import** it. Importing a module is done by an instruction named ***import***.
- Note: import is also a keyword (with all the consequences of this fact).



- Let's assume that you want to use two entities provided by the math module:
  - a symbol (constant) representing a precise value of π
  - a function named sin()
- Both these entities are available through the math module, but the way in which you can use them strongly depends on how the import has been done.

- The simplest way to import a particular module is to use the import instruction as follows:

```
import math
```

- If you want to (or have to) import more than one module, you can do it by repeating the `import` clause (preferred):

```
import math
import sys
```

- or by listing the modules after the `import` keyword, like here:

```
import math, sys
```

- The instruction imports two modules, first the one named `math` and then the second named `sys`.

- CONTENTS
- Outcomes
- What is a module?
- Using a Module
- **IMPORTING A MODULE**

```
import math


def sin(x):
    if 2 * x == pi:
        return 0.99999999
    else:
        return None


pi = 3.14

print(sin(pi/2))
print(math.sin(math.pi/2))
```


- Exercise: Perform the following tasks
- Use from keyword to import pi from the math module
- Import e (the mathematical constant) from the math module.
- Write a program to perform the following operation sin(pi/2)

- Exercise: Perform the following tasks
- Use from keyword to import pi from the math module

```
from math import pi
```

- Import e (the mathematical constant) from the math module.

```
import math
Print(math.e)
```

- Write a program to perform the following operation sin(pi/2)

```
from math import sin, pi
print(sin(pi/2))
```

- CONTENTS
- Outcomes
- What is a module?
- Using a Module
- **IMPORTING A MODULE**

```
pi = 3.14


def sin(x):
    if 2 * x == pi:
        return 0.99999999
    else:
        return None


print(sin(pi / 2))

from math import sin, pi

print(sin(pi / 2))
```

- Importing a module: *
  - The following import's syntax is a more aggressive form of the previously presented one:

```
from module import *
```

  - As you can see, the name of an entity (or the list of entities' names) is replaced with a single asterisk (*).
  - Such instruction imports all entities from the indicated module.
  - Is it convenient? Yes, it is, as it relieves you of the duty of enumerating all the names you need.
  - Is it unsafe? Yes, it is - unless you know all the names provided by the module, you may not be able to avoid name conflicts. Treat this as a temporary solution, and try not to use it in regular code.

- Importing a module: the as keyword
  - If you use the import module variant and you don't like a particular module's name (e.g., it's the same as one of your already defined entities, so qualification becomes troublesome) you can give it any name you like - this is called aliasing.
  - Aliasing causes the module to be identified under a different name than the original. This may shorten the qualified names, too.
  - Creating an alias is done together with importing the module, and demands the following form of the import instruction:

```
import module as alias
```

  - The "module" identifies the original module's name while the "alias" is the name you wish to use instead of the original.
- Note: as is a keyword.

- Examples 1

```
import math as m
print(m.sin(m.pi/2))
```

- Example 2

```
from math import pi as PI, sin as sine
print(sine(PI/2))
```