MODULES & PACKAGES

IN PYTHON PROGRAMMING

MODULES

- A python modules can be defined as a python program file.
  - It contains – functions, class or variables.
- Modules in Python provide us the flexibility to organize the code in a logical way.
- Example:

```
def my Module(name)
        print(“Hi!”, name);
```

- Save the above code with <filename>.pie
- **programs are designed to be run**, whereas **modules are designed to be imported** and **used by programs**.

MODULES

- **Modules** - a module is a piece of software that has a specific functionality. Each module is a **different file**, which can be edited separately.
- **Modules** provide a means of **collecting sets of Functions together** so that they can be **used by** any **number of programs.**

- Module is like a code library which can be used to borrow code written by somebody else in our python program. There are two types of modules in python:

1. **Built in Modules** - These modules are ready to import and use and ships with the python interpreter. there is no need to install such modules explicitly.

2. **External Modules -** These modules are imported from a third party file or can be installed using a package manager like pip or conda. Since this code is written by someone else, we can install different versions of a same module with time.

THE PIP COMMAND

It can be used as a package manager [pip](https://pip.pypa.io/en/stable/) to install a python module.

- Lets install a module called pandas using the following command
-
- pip install pandas

LOADING A MODULE IN PYTHON CODE

- Python provides 2 types of statements:
  - Import Statement
  - From-import Statement

IMPORT STATEMENT

- Import statement is used to import all the functionality of one module into another.
- We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times.
- Syntax:

```
import module1, modeule2, . . . . , module n
```

- Example:

```
import file
name = input(“Enter your name:”)
print(name)
```

- We can **import a module** using the **import statement** and **access** the **definitions** inside it using **the dot operator**.
- import math
- print("The value of pi is", Math.pi)

Import with renaming

- We can import a module by renaming it as follows:
- # import module by renaming it
- import math as m
- print("The value of pi is", Math.pi)

FROM-IMPORT STATEMENT

- It provides the flexibility to import only the specific attributes of a module.
- Syntax:

```
from <module-name> import <name1>, <name2> . . .
```

- Calculation. Pie

```
def sum(am):
        return a + b
def mul(a,b):
        return a * b
def div(a,b):
        return a/b
```

FROM-IMPORT STATEMENT

- We can **import specific names** from a module **without importing the module as a whole**. Here is an example.

- # import only pi from math module
- from math import pi
- print("The value of pi is", pi)

IMPORT ALL NAMES

- We can import all names(definitions) from a module using the following construct:

- from math import *
- print("The value of pi is", pi)

EXAMPLE

```
from calculation import sum          #it will import only the sum() from calculation. Pie
a = int(input(“Enter the first number”))
b = int(input(“Enter the second number”))
print(“Sum = “, sum(a,b))
```

IMPORT AS STATEMENT [RENAMING A MODULE]

- Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in python source file.
- Syntax:

```
import <module-name> as <specific-name>
```

- Example:

```
import calculation as call;
a = int(input(“Enter first number”))
b = int(input(“Enter second number”))
print(“Sum = “, call. Sum(a,b))
```

DIR( ) FUNCTION

- The dir.( ) function returns a sorted list of names defined in the passed module.
- This list contains all the sub-modules, variables and functions defined in this module.
- Example:

```
import Json
list = dir.(Json)
print(list)
```

DIR( ) FUNCTION

- We can use the dir.() function to find out names that are defined inside a module.
- we have defined a function add() in the module example that we had in the beginning.
- dir.(example)
- ['__built-ins__', '__cached__', '__doc__', '__file__', '__initializing__', '__loader__', '__name__', '__package__', 'add']

- a sorted list of names (along with add).
- All other names that begin with an **underscore** are **default Python attributes associated with the module** (not-user-defined).

LET US CREATE A MODULE

- Type the following and save it as **example. Pie**.
- # Python Module example
- def add(a, b):
  - """This program adds two numbers and return the result"""
  - result = a + b
  - return result

HOW TO IMPORT MODULES IN PYTHON?

```
import example


example.Add(4,5.5)
9.5  # Answer
```

VARIABLES IN MODULE

- The module can **contain functions**, as already described, but also **variables** of all types (arrays, dictionaries, objects etc.):
- Save this code in the file my module. Pie

-
  ```
  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```

VARIABLES IN MODULE

- Import the module named my module, and access the person1 dictionary:
- import my module

  a = my module.person1["age"]
  print(a)
- Run Example → 36
-

IMPORT FROM MODULE

- The module named **my module** has one function and one dictionary:

-
  ```
  def greeting(name):
    print("Hello, " + name)

  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```
- Example
- Import only the person1 dictionary from the module:

-
  ```
  from my module import person1

  print (person1["age"])
  ```

IMPORT FROM MODULE

-
  ```
  def greeting(name):
    print("Hello, " + name)

  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```

- Example
- Import all objects from the module:

-
  ```
  from my module import *
  print(greeting(“Ram”)
  print (person1["age"])
  ```

BUILT-IN MODULES

- Python comes with a rich standard library of **built-in modules** that provide a wide range of functionality.
- These modules are pre-installed with Python and do not require additional installation.
- They can be imported into your programs using the import statement.

1- os(Operating System Interfaces)-

Provides functions to interact with the operating system, such as file manipulation and environment variables.

Example-

```
import os

# Get the current working directory
print(os.getcwd())

# List files in the current directory
print(os.listdir())

# Create and remove a directory
os.mkdir('test_dir')
os.rmdir('test_dir')
```

2. sys (System-Specific Parameters and Functions)

Provides access to system-specific parameters and functions

Example-

```
import sys

# Get the Python version
print(sys.version)

# Command-line arguments
print(sys.argv)

# Exit the program
sys.exit("Exiting the program")
```

Output-

```
3.11.3 (tags/v3.11.3:f3909b8,Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)]
['d:\\dir.\\d']
Exiting the program
```

3. math (Mathematical Functions)

Provides mathematical operations and constants.

Example-

```
import math

# Compute square root
print(math.sqrt(16))

# Use constants
 print(math.pi)
 print(math.e)

# Trigonometric functions
print(math.sin(math.radians(30)))
```

Output-

```
4.0
3.141592653589793
2.718281828459045
0.49999999999999994
```

4. **random** (Generate Random Numbers)-
Generates random numbers and makes random selections.

Example-

```
import random

# Generate a random number between 0 and 1
print(random.random())

# Generate a random integer between 1 and 10
print(random.randint(1, 10))

# Shuffle a list
numbers = [1, 2, 3, 4, 5]
random.shuffle(numbers)
print(numbers)
```

Output-

```
0.4879269052859423   |  0.4583495829674801
7                    |   3
[2, 4, 1, 5, 3]      |   [5, 2, 4, 3, 1]
```

5. datetime (Date and Time Handling)

Provides functions to manipulate dates and times.

Example-

```
import datetime

# Get the current date and time
now = datetime.datetime.now()
print(now)

# Format the date
print(now.strftime('%Y-%m-%d %H:%M:%S'))

# Calculate a future date
future = now + datetime.timedelta(days=5)
print(future)
```

Output-

```
2024-11-17 07:44:08.944251
2024-11-17 07:44:08
2024-11-22 07:44:08.944251
```

6. **Json** (JSON Data Handling)

Handles JSON data for reading and writing.

Example-

```
import json

# Convert a dictionary to a JSON string
data = {"name": "John", "age": 30}
json_data = json.dumps(data)
print(json_data)

# Convert a JSON string back to a dictionary
parsed_data = json.loads(json_data)
print(parsed_data)
```

7. **re** (Regular Expressions)-
Provides tools for pattern matching and text searching

Example-

```
import re

# Check if a string contains a pattern
pattern = r'\d+'
text = "There are 123 apples"
match = re.search(pattern, text)
if match:
    print("Found:", match.group())
```

Output-

```
Found: 123
```

Counter, deque, and defaultdict.

8. collections (Specialized Data Structures)

Provides specialized container types such as Counter, deque, and defaultdict

Example-

```
from collections import Counter

# Count occurrences of elements
data = ['a', 'b', 'a', 'c', 'b', 'a']
counter = Counter(data)
print(counter)
```

Output-

```
Counter({'a': 3, 'b': 2, 'c': 1})
```

9. itercools (Iterator Tools)

Offers functions for creating and working with iterators.

Example-

```
import itertools

# Generate permutations
perms = itertools.permutations([1, 2, 3])
print(list(perms))

# Infinite counting
for i in itertools.count(10, 2):
    if i > 20:
        break
    print(i)
```

Output-

```
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
10
12
14
16
18
20
```

10. time (Time Access and Conversions)

Provides time-related functions

Example-

```
import time

# Get the current time
print(time.time())

# Sleep for 2 seconds
time.sleep(2)
print("Slept for 2 seconds")
```

RELOAD( ) FUNCTION

- If you want to reload the already imported module to re-execute the top-level code, python provides us the reload( ) function.
- Syntax:

```
 import importlib
importlib. reload(module name)
```

- The reload() function in Python is used to reload a previously imported module. This can be helpful during development when you modify a module and want to see the changes without restarting the interpreter.
- The reload() function is part of the importlib module in Python 3. It allows you to re-execute the module's code.

Example:

Before Reloading

```
# File: my module. Pie
def greet():
    print("Hello!")

# Main script
import mymodule
mymodule.greet() # Output: Hello!
```

Modify the Module (my module. Pie)

```
# File: mymodule.py (modified)
 def greet():
   print("Hello, World!")
```

```
# Main script
import importlib

# Reload the module
importlib.reload(mymodule)

# Test the updated function
mymodule.greet()  # Output: Hello, World!
```

PYTHON PACKAGES

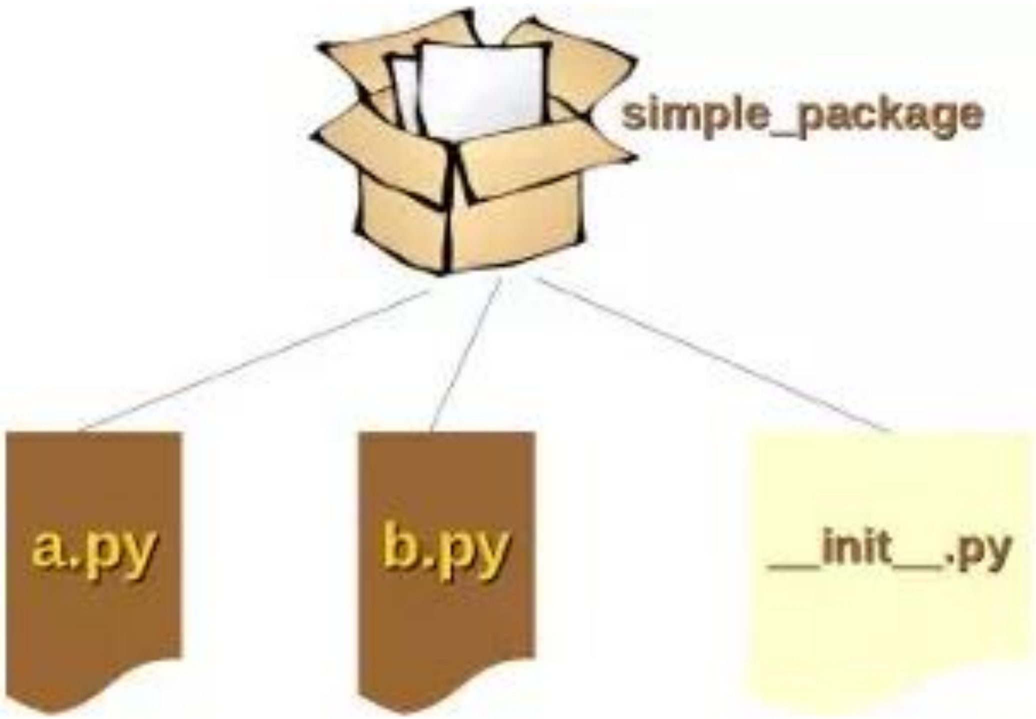
- The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules.
- The packages are used to categorize the application level code efficiently.

- We don't usually store all of our files on our computer in the same location.
- We use a **well-organized hierarchy** of directories for easier access.
- Similar **files** are kept in the **same directory**, for example, we may keep **all the songs** in the "**music**" directory.
- similar to this, Python has packages for directories and modules for files

PYTHON PACKAGES

- As our application program grows **larger in size with a lot of modules**, we place **similar modules in one package** and different modules in different packages.
- This makes **a project (program) easy to manage** and **conceptually clear**.
- Similarly, **as a directory can contain subdirectories and files**, a **Python package can have sub-packages and modules**.

A SIMPLE EXAMPLE OF PYTHON PACKAGE

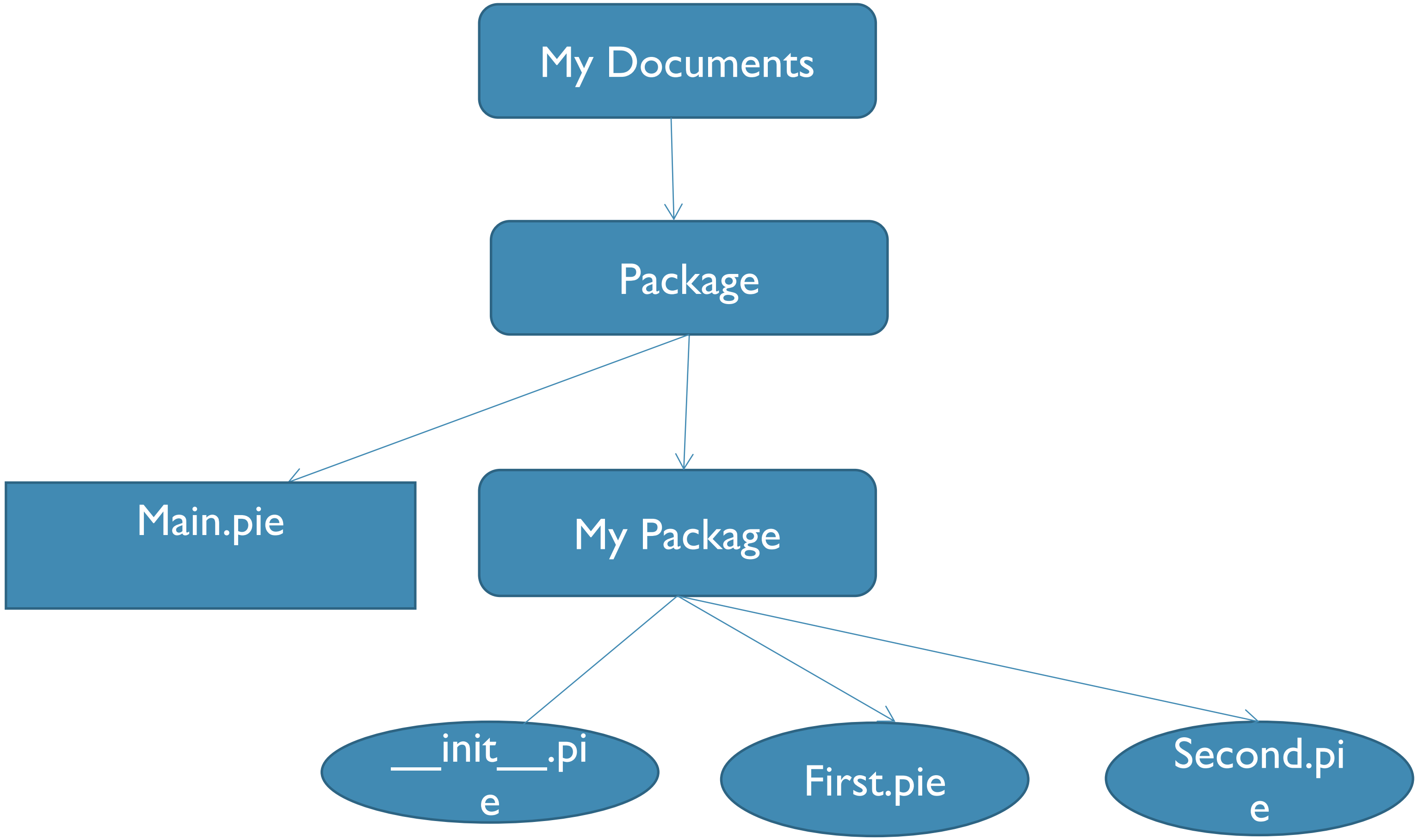


PYTHON PACKAGES

- First of all, **we need a directory**. The name of this directory will be the name of the package, which we want to create.
- We will **call our package "simple package". This directory needs to contain a file with the name __init__.pie**.
- This file can be empty, or it can contain valid Python code.
- This code will be **executed when a package is imported**, so it can be used to initialize a package.
- We create two simple files a.pie and b.pie just for the sake of filling the package with modules

__INIT__.PIE

- A **directory must contain a file named __init__.pie** in order for Python to consider it as a package.
- This file can be left empty but we generally place the initialization code for that package in this file


My Documents
Package
Main.pie
My Package
__init__.pie
First.pie
Second.pie

First.py

```
    def one():
        print(“First Module”)
        return
```

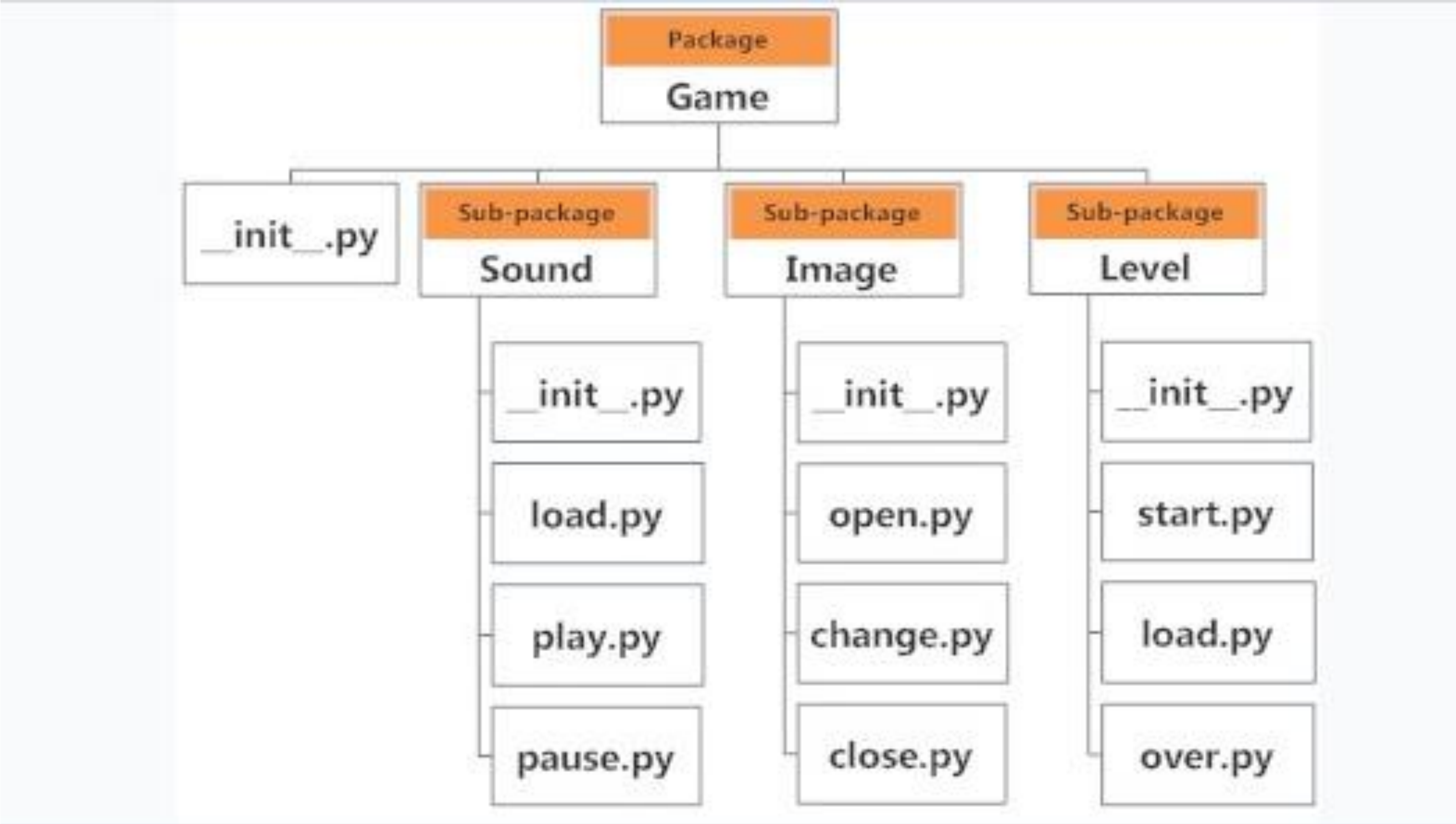
Second.py

```
    def second():
        print(“Second Module”)
        return
```

Main.py

```
from My-Package import First
First.one()


from My-Package import First, Second
First.one()
Second. Second()
```



Package Module Structure in Python Programming

- For example, if we want to **import the start module** in the above example, it can be done as follows:

```
import Game.Level.start
```

- Now, if this **module contains a function named select difficulty()**, we must use the full name to reference it.

```
Game.Level.start.select_difficulty(2)
```

- If this construct seems lengthy, we can **import the module without the package** prefix as follows:

```
from Game.Level import start
```

- We can now call the function simply as follows:

```
start.select_difficulty(2)
```

- Another way of importing just the required function (or class or variable) from a module within a package would be as follows:

```
from Game.Level.start import select_difficulty
```

- Now we can directly call this function.

```
select_difficulty(2)
```