UNIT 2: FILE, EXCEPTION HANDLING AND OOP

1. **User defined Modules and Packages in Python**
2. **Files:**
   a) **File manipulations,**
   b) **File and Directory related methods**
3. **Python Exception Handling.**
4. **OOPs Concepts**
5. **Class and Objects**
6. **Constructors**
7. **Data hiding**
8. **Data Abstraction**
9. **Inheritance**

Modular Programming

- Modular programming refers to **the process of breaking a large programming task (code) into separate, smaller, more manageable subtasks** called as **modules**.

Modules

- **To create an application, the entire program is divided into sub parts which is called as Modules.**
- **In Python, a file that contains a set of code to perform any task** is called a Module.
- This set of code can be **classes, functions, variables, etc.**
- These codes **can be reused in multiple other Python programs**.

For example: A file **Program.py is called a module**, and its module name would be **Program.**

With the help of modules, we can organize related functions, classes, or any code block in the same file. So, it is considered a best practice Data Science is to split the large Python code blocks into modules containing up to 300–400 lines of code.

Advantages of using modules:

- **Simplicity:** a module typically focuses on one relatively small portion of the problem.
- **Maintainability:** modifications to a single module will have an impact on other parts of the program.
- **Reusability:** Functionality defined in a single module can be easily reused by other parts of the application.
- **Scoping:** Modules avoid collisions between identifiers in different areas of a program.
- **Structuring:** Grouping related code into a module makes the code logically organized.

Types of Modules

Modules in Python can be categorized into two types:

1) Built-in Modules
2) User-defined Modules

1. Built-in Modules in Python

- One of the important feature of Python is that it comes with a “rich standard library”.
- This rich standard library contains lots of built-in modules which provides a lot of reusable code.
- User can import and use any of the built-in modules in a program.

Example: Python contains Built-in Modules like “math”, “random”, “datetime”, etc.

Example (To use built-in math module):

```
import math # importing built-in module math
```

```
print(math.sqrt(25)) # using square root(sqrt) function contained in math module

print(math.pi) # using pi function contained in math module

print(math.factorial(4))  #using factorial function in math module
```

2. User-defined Modules

- When a user wants to create an application, the large program can be separated into smaller parts.
- **This small part is a file created by the user that contains a set of code for any task that the user wants to include in an application** is called a User-defined Module.

  In other words, a user creates own functions and classes, put them inside a file which is called as user-defined modules.

a) Creating a Module (How to create a Module?)

- When a user writes a code in a file and saves it with any name and using the file extension .py then this file is called as Module and the module is said to be created.
- **Naming a Module**
  - You can name the module file whatever you like, but it must have the file extension .py
- **Variables in Module**
  - The module can contain classes, functions or variables of all data types (arrays, dictionaries, objects etc):

Example:

```
# Calc.py is a simple module we are creating with two functions inside the file
 def add(x, y):
  return (x+y)
 def subtract(x, y):
  return (x-y)
```

b) Using a Module/ (How to use a Module in any Program?) (import statement)

- To use a Module(functions and classes defined in a file) in any other program, **import** statement is used.
- **Syntax:**

  Import module_name
- The import keyword with the module name is used to import a Module.
- **Different methods of Using/ importing modules are**:

  a) Renaming a Module (as statement)
  - We can rename the module while importing it using the **as** keyword.
  - It allows the user to give a shorter name to a module while using it in your program.
  - **Syntax:**

    import <module_name> as new_name

- **Example:**

  # Renaming calc module using as keyword.

  >>import calc as c

  >>c.add(2,4)

b) Importing items from a Module (from statement)

- It allows the user to import specific attributes from a module **without importing the module as a whole.**
- **Syntax:**

  from <modulename> import <**module_attributes**>

- **Example:**

>>from calc import add

>>add(20,20)

c) Importing multiple attributes

- Multiple attributes and functions can be imported from a module using from statement by separating the attribute names by using comma (,)
- **Syntax:**

  from module_name import x, y, z, a
- **Example:**

  >>> from calc import add, subtract

  >>>print(add(10,10))

  >>>print(subtract(20,10)

d) Import all items from a Module (using * statement)

- The * symbol used with the from import statement is used to import all the names from a module to a current namespace.
- **Syntax:**

  from module_name import *

- **Example:** Importing all names

  >>from calc import *

  >>print(add(5,10))

  >>print(subtract(15,12)

Note:

✓ **While importing from a module using from statement, the user do not need to use the dot operator while calling the function or using the attribute.**

✓ **To access the functions inside the module the dot(.) operator is used.**

importing module calc.py

import calc

print(calc.add(10, 2))

c) dir() function

- The dir() built-in function **returns a sorted list of strings containing the names of all the modules, variables, and functions defined by a module**.
- **Example:**

  The __name__ attribute contains the name of the module.

  >>> import calc

  >>>dir(calc)

d) Reloading a Module

- Suppose you have already imported a module and using it.
- However, the owner of the module added or modified some functionalities after you imported it.
- So, you can reload the module to get the latest module using the reload() function, as shown below.

  >>>import calc

  >>>reload(calc)

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.

- The Package **is a simple directory or folder having collections of sub-packages and modules as files.**
- The Python interpreter **recognizes a folder as the package if it contains __init__.py file.**
- __init__.py serrves the following:
  - ✓ __init__.py file **specifies the resources (functions) from its modules to be imported.**
  - ✓ **An empty __init__.py file makes all functions from the modules available when this package is imported.**
- **Example:**

  Student(Package)

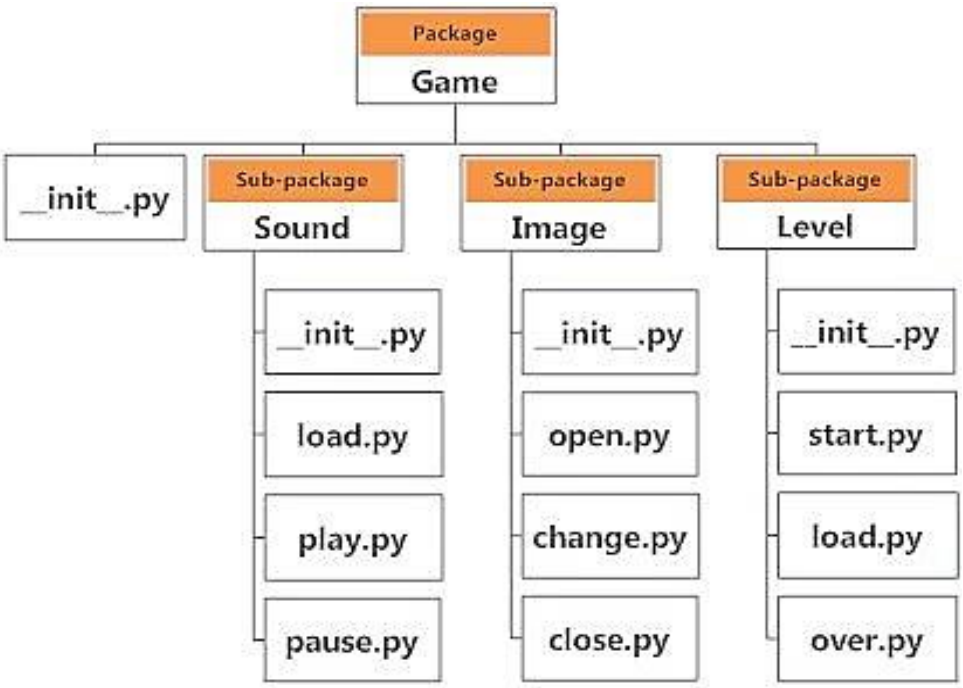
  | __init__.py (Constructor)

  | details.py (Module)

  | marks.py (Module)

  | collegeDetails.py (Module)

Creating Package



- Create a package named Game.
- Inside this folder create three sub-folder for three sub-packages Sound, Image, Level.
- Inside each folder create an empty Python file i.e. __init__.py
- Then create three Modules in each sub package..

Example: Importing Modules from a Package

- We can import these modules using the from…import statement and the dot(.) operator.
- While importing a package or sub packages or modules, Python searches the whole tree of directories looking for the particular package and proceeds systematically as programmed by the dot operator.

  Syntax:
  import package_name.module_name

  Example:
  import Games.Sound,play

  OR

  from Games.Sound import play

  OR

  from Games.Sound import *

  This will import everything i.e., modules, sub-modules, function, classes, from the sub-package.

Note:

Library: The **library** is having a collection of related functionalities of codes that allows you to perform many tasks without writing your code.

It is a reusable chunk of code that we can use by importing it in our program, we can just use it by importing that library and calling the method of that library with period(.).

Example:

Importing Pandas library and call read_csv method using alias of pandas i.e. pd.

```
import pandas as pd
df = pd.read_csv("file_name.csv")
```