Introduction

We have some rules while writing programs in any programming language, such as not using a space when defining a variable name, adding a colon (:) after the if statement, and so on. If we don’t follow these rules, we run into Syntax Errors and our program refuses to execute until we squash those errors.

But there are occasions when the program is syntactically correct and it still throws up an error when we try to execute the program. Well, these errors detected during the execution are called exceptions. And dealing with these errors is called exception handling.

Advantages of Exception Handling

1. Let’s say you have written a script to read thousands of files present in multiple directories. Now, there can be some sort of error there, such as a file type is missing or has an incorrect format or it is available in different extensions. In that case, it is not feasible to open all the files and write a script accordingly. **Exception handling allows us to define multiple conditions**. For example, if the format is wrong, correct the format first and then try to read the file. Otherwise, skip reading that file and create a log file so that we can deal it with later
2. Second, let’s say we are scraping restaurant data from a website and our script looks for the name, reviews and the address of the restaurant. Due to some reason, the address of the restaurant is missing from the website. In this case, if we are not handling the exceptions, our script can stop in between so while collecting data in a huge amount it is essential that we handle the exceptions

Common Exceptions

- **ZeroDivisionError**: It is raised when you try to divide a number by zero
- **ImportError**: It is raised when you try to import the library that is not installed or you have provided the wrong name
- **IndexError**: Raised when an index is not found in a sequence. For example, if the length of the list is 10 and you are trying to access the 11th index from that list, then you will get this error
- **IndentationError**: Raised when indentation is not specified properly
- **ValueError**: Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified
- **Exception**: Base class for all exceptions. If you are not sure about which exception may occur, you can use the base class. It will handle all of them.

What are Python Modules?

A module is a pythonic statement which contains multiple functions in it. Modules act as a pre-defined library in the code, which is accessible to both coder and user.
The python modules also store pre-defined functions from the library while the execution of code is going on.

What are Python Packages?

A package is the form of a collection of tools which helps in the initiation of the code. A python package acts as a user-variable interface for any source code. This makes a python package work at a defined time for any functionable code in the runtime.

We will demonstrate with a very simple example how to create a package with some Python modules. 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__.py. 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, e.g. to make sure that some other modules are imported or some values set. Now we can put all of the Python files which will be the submodules of our module into this directory. We create two simple files a.py and b.py just for the sake of filling the package with modules.

Differences Between Python Modules and Packages

1. A package holds the file __init__.py for every user-oriented code. But this does not apply to modules in runtime for any user-specific codes.
2. A module is a file containing Python code in run time for a user-specific code. A package also modifies the user interpreted code in such a way that it gets easily functioned in the run time.

Python Inheritance

Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more.

Inheritance¶

Inheritance allows us to define a class that inherits all the methods and attributes from another class. Convention denotes the new class as **child class**, and the one that it inherits from is called **parent class** or **superclass**. If we refer back to the definition of class structure, we can see the structure for basic inheritance is **class ClassName(superclass)**, which means the new class can access all the attributes and methods from the superclass. Inheritance builds a relationship between the child class and parent class, usually in a way that the parent class is a general type while the child class is a specific type.

Encapsulation¶

Encapsulation is one of the fundamental concepts in OOP. It describes the idea of restricting access to methods and attributes in a class. This will hide the complex details from the users, and prevent data being modified by accident. In Python, this is achieved by using private methods or attributes using underscore as prefix, i.e. single “_” or double “__”. Let us see the following example.