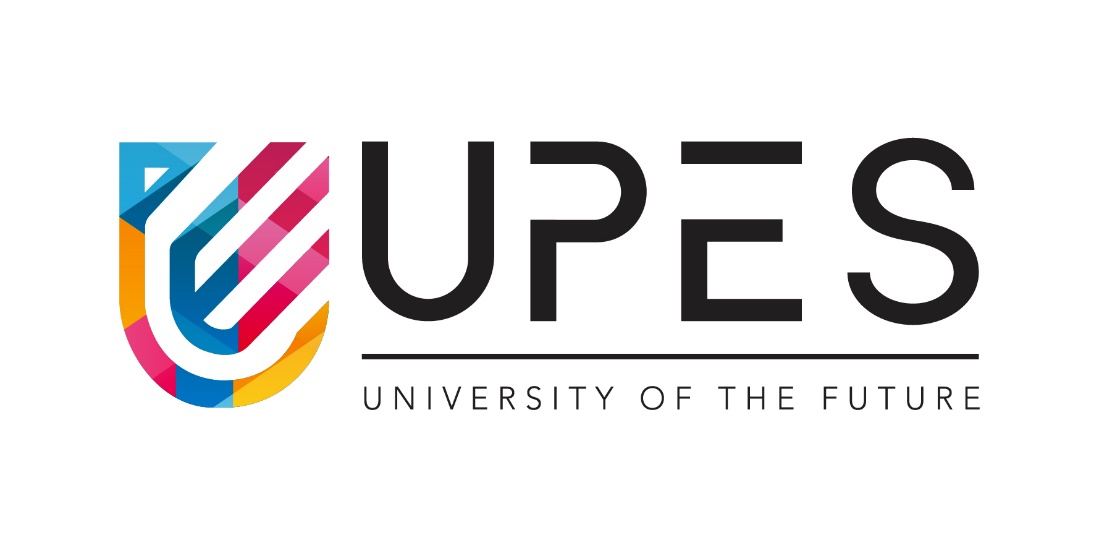
**ONLINE LEARNING OUTCOME GROUP PROJECT**



B.TECH. CSE CCVT 5th Semester

GROUP MEMBERS:

1. Aayush Chowdhary(SAP : 500086126)
2. Anas Khan(SAP : 500086206)
3. Nilesh Verma(SAP : 500087239)
4. Ashish Gusain(SAP : 500087234)
5. Vikrant Kaushik (SAP : 500083739)
6. Chinmay Bhura(SAP : 500085560)
7. Diwanshu Chandila (SAP : 500086142)

***CONTENT AGGREGRATOR:***

A content aggregator is an entity that pulls together web or media content, applications or both from online sources for reuse or resale. It's a means of curating content. Two types of content aggregators exist: those who gather news and other materials from various sources for publication on their own Web sites.

Content Aggregator is a powerful tool when we want to save time and accurate results.

We can say that content aggregator works as a crawling mechanism and also as a scraping tool.

Our team used Python language for the execution of this project.

Python has various libraries that were helpful in creating the aggregator. We will also discuss the various libraries and various OOPs and Python Programming Concepts used in this project.

Given below are the libraries and OOPs concepts that were used in our Content Aggregator.

**BEAUTIFULSOUP:**

[Beautiful Soup](http://www.crummy.com/software/BeautifulSoup/) is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. It commonly saves programmers hours or days of work.

Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. But you’ll only ever have to deal with about four *kinds* of objects: Tag, NavigableString, BeautifulSoup, and Comment.

**REQUEST:**

The [requests](http://docs.python-requests.org/en/master/) library is the de facto standard for making HTTP requests in Python. It abstracts the complexities of making requests behind a beautiful, simple API so that you can focus on interacting with services and consuming data in your application.

Throughout this article, you’ll see some of the most useful features that requests have to offer as well as how to customize and optimize those features for different situations you may come across. You’ll also learn how to use requests in an efficient way as well as how to prevent requests to external services from slowing down your application.

**PYTHON-DOCX:**

*python-docx* is a Python library for creating and updating Microsoft Word (.docx) files.

**CLASSES in Python:**

Classes provide a means of bundling data and functionality together. Creating a new class creates a new *type* of object, allowing new *instances* of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

Compared with other programming languages, Python’s class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object-Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

**PARSER MODULE:**

The [parser](https://docs.python.org/3.9/library/parser.html#module-parser) module provides an interface to Python’s internal parser and byte-code compiler. The primary purpose for this interface is to allow Python code to edit the parse tree of a Python expression and create executable code from this. This is better than trying to parse and modify an arbitrary Python code fragment as a string because parsing is performed in a manner identical to the code forming the application. It is also faster.

**DATA ENCAPSULATION:**

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object’s variable can only be changed by an object’s method. Those types of variables are known as **private variables.**

**DATA ABSTRACTION:**

Abstraction is really powerful for making complex tasks and codes simpler when used in Object-Oriented Programming. It reduces the complexity for the user by making the relevant part accessible and usable, leaving the unnecessary code hidden. Also, there are times when we do not want to give out sensitive parts of our code implementation and this is where data abstraction can also prove to be very functional.

Data Abstraction in Python can be achieved through creating abstract classes and inheriting them later. Before discussing what, abstract classes are, let us have a brief introduction of inheritance.

Inheritance in OOP is a way through which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class/subclass.

**INHERITANCE:**

Python is an Object-Oriented Programming language and one of the features of Object-Oriented Programming is Inheritance. Inheritance is the ability of one class to inherit another class. Inheritance provides reusability of code and allows us to create complex and real-world-like relationships among objects.

The class which got inherited is called a parent class or superclass or base class.

The class which inherited the parent class is called a child class or subclass or derived class or extended class.