|  |  |
| --- | --- |
| pencil and paper icon | Assignment  Algorithms & Python Basics  By: Ammar Fadhel |

|  |  |
| --- | --- |
| How to download videos using web scraping | |
|  | To download videos using web scraping in Python, you can follow the steps below: |
| ☐ | Install and import the following libraries in your Python environment:  ● requests: For making HTTP requests to retrieve web page content.  ● BeautifulSoup: For parsing and extracting information from HTML pages. |
| ☐ | Use the requests library to send an HTTP GET request and retrieve the web page that contains the video. |
| ☐ | Use BeautifulSoup to parse the HTML page and extract relevant elements such as video tags. |
| ☐ | Analyze the HTML page and extract the links associated with the videos. Video links can be found in elements like <video>, <source>, or any other element that contains the video link. |
| ☐ | Use requests to send HTTP requests to download the videos from the extracted links. You can use a function like requests.get() to download the video and save it to your computer. |

|  |  |
| --- | --- |
| SOLID princeples | |
|  | The SOLID principles are a set of five design principles that aim to make software systems more maintainable, flexible, and scalable. These principles were introduced by Robert C. Martin (also known as Uncle Bob) and have become widely adopted in the field of software development. Let's go through each principle:  ● **Single Responsibility Principle (SRP):**  The SRP states that a class or module should have only one reason to change. In other words, a class should have a single responsibility or purpose. By adhering to this principle, you can ensure that classes are focused and have a clear and well-defined responsibility, making them easier to understand, test, and maintain.  ● **Open/Closed Principle (OCP):**  The OCP states that software entities (classes, modules, functions) should be open for extension but closed for modification. This means that you should be able to extend the behavior of a system without modifying its existing code. By relying on abstractions, interfaces, and polymorphism, you can introduce new functionality by adding new classes or modules instead of modifying the existing ones, minimizing the risk of introducing bugs or breaking existing code.  **● Liskov Substitution Principle (LSP):**  The LSP states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. In other words, derived classes must be substitutable for their base classes, preserving the behavior expected by the clients. Adhering to this principle ensures that inheritance hierarchies are well-designed, promoting reusability and avoiding unexpected side effects.  ● **Interface Segregation Principle (ISP):**  The ISP states that clients should not be forced to depend on interfaces they do not use. It emphasizes the idea of creating fine-grained and specific interfaces tailored to the needs of clients. By avoiding fat interfaces and splitting them into smaller, focused ones, you prevent clients from being burdened with unnecessary dependencies, reducing coupling and making the system more maintainable and cohesive.  ● **Dependency Inversion Principle (DIP):**  The DIP states that high-level modules should not depend on low-level modules; both should depend on abstractions. This principle promotes loose coupling by inverting the traditional dependency flow. Instead of depending on concrete implementations, components should depend on abstractions (interfaces or abstract classes). This allows for interchangeable and easily replaceable implementations, enabling flexibility, testability, and modularity. |

|  |  |
| --- | --- |
| Binary Search | |
| ☐ | **Pseudocode:** |
| ☐ | **Flow Chart**: |

|  |  |
| --- | --- |
| lambda functions | |
|  | Lambda functions are typically used in situations where you need a small, one-time function without explicitly defining a named function.  Here's an example of a lambda function in Python that doubles a given number:    In this example, lambda x: x \* 2 defines a lambda function that takes an argument x and returns its doubled value. The lambda function is assigned to the variable double. You can then call the lambda function like any other function. |