**Web scraping:**

To scrape YouTube videos using Python, you can use the `youtube-dl` library. Here's an example code snippet:

import youtube\_dl

def scrape\_youtube\_video(url):

ydl\_opts = {

'format': 'best',

'ignoreerrors': True,

'postprocessors': [{

'key': 'FFmpegVideoConvertor',

'preferedformat': 'mp4'

}],

}

with youtube\_dl.YoutubeDL(ydl\_opts) as ydl:

info\_dict = ydl.extract\_info(url, download=False)

video\_title = info\_dict.get('title', None)

video\_url = info\_dict.get('url', None)

print(f"Title: {video\_title}")

print(f"URL: {video\_url}")

Make sure you have `youtube-dl` :

installed 'pip install youtube-dl'.

**1- SOLID principles:**

The SOLID principle is a set of five design principles that help software developers design and implement software systems that are easy to understand, maintain, and extend. These principles were introduced by Robert C. Martin (also known as Uncle Bob) and have become widely adopted in the software development industry.

The five SOLID principles are:

1. Single Responsibility Principle (SRP): A class should have only one reason to change. This principle states that a class should have only one responsibility or job, and it should not be responsible for multiple unrelated tasks. By adhering to this principle, classes become more focused, easier to understand, and less likely to require modification when changes occur.

2. Open/Closed Principle (OCP): Software entities (classes, modules, functions) should be open for extension but closed for modification. This principle encourages developers to design their code in a way that allows new functionality to be added without modifying existing code. This is typically achieved through the use of abstractions, interfaces, and inheritance.

3. Liskov Substitution Principle (LSP): Subtypes must be substitutable for their base types without affecting the correctness of the program. In simpler terms, this principle states that objects of a superclass should be able to be replaced with objects of its subclass without causing any issues or breaking the program's behavior.

4. Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use. This principle emphasizes that interfaces should be specific and tailored to the needs of clients who use them. It discourages creating large interfaces with many methods when clients only need a subset of those methods.

5. Dependency Inversion Principle (DIP): High-level modules should not depend on low-level modules; both should depend on abstractions. This principle promotes loose coupling between modules by introducing abstractions or interfaces between them. It allows for easier testing, flexibility in changing implementations, and better separation of concerns.

By following these SOLID principles, developers can create software that is more modular, maintainable, and scalable.

**2- Lambda function in python:**

Lambda functions in Python are anonymous functions that can be defined without a name. They are typically used when you need a small, one-line function without the need for a full function definition.

The syntax for a lambda function is as follows:

lambda arguments: expression

Here, 'arguments' represent the input parameters of the function, and 'expression' is the computation or operation to be performed on those arguments.

For example, let's say we want to define a lambda function that takes two numbers as input and returns their sum:

add = lambda x, y: x + y

We can then call this lambda function like any other regular function:

result = add(5, 3)

print(result) # Output: 8

Lambda functions are often used in conjunction with built-in functions like 'map()', 'filter()', and 'reduce()' to perform operations on iterables in a concise manner.

[[1]](#footnote-1)

**3- Binary search algorithm:**

The binary search algorithm is an efficient searching algorithm used to find a specific target value within a sorted array or list. It follows the divide and conquer approach by repeatedly dividing the search space in half until the target value is found or determined to be not present.

Here is how the binary search algorithm works:

1. Start with defining the lower and upper bounds of the search space, which are initially set to the first and last indices of the array, respectively.

2. Calculate the middle index of the search space by taking the average of the lower and upper bounds:

'mid = (lower + upper)/2'.

3. Compare the target value with the element at the middle index:

- If they are equal, return the middle index as it is the position of the target value.

- If the target value is less than the middle element, update the upper bound to 'mid - 1' and go back to step 2.

- If the target value is greater than the middle element, update lower bound to 'mid + 1'and go back to step 2.

4. Repeat steps 2-3 until either:

- The target value is found and returned.

- The lower bound becomes greater than or equal to upper bound, indicating that the target value does not exist in the array.

The binary search algorithm has a time complexity of O(log n), where n is the number of elements in the array. This makes it significantly faster than linear search for large arrays.

It's important to note that for binary search to work correctly, it requires a sorted array as input. If your array is not sorted, you may need to sort it first using another sorting algorithm like quicksort or mergesort before applying binary search.

1. [↑](#footnote-ref-1)