PYTHON LIST

Python List

파이썬 리스트

내장된 동적 크기 배열(자동으로 커지고 줄어듬)입니다.
모든 유형의 항목(다른 리스트 포함)을 리스트에 저장할 수 있습니다.
리스트는 혼합된 유형의 항목을 포함할 수 있는데, 이는 리스트가 주로 인접한 위치에 참조를 저장하고 실제 항목은 다른 위치에 저장될 수 있기 때문에 가능합니다.

- 목록에는 중복된 항목이 포함될 수 있습니다.
- 파이썬의 리스트는 변경 가능합니다. 따라서 항목을 수정, 대체 또는 삭제할 수 있습니다.
- 목록은 정렬됩니다. 요소가 추가되는 방식에 따라 요소의 순서를 유지합니다.
- 목록에 있는 항목에 접근하려면 0부터 시작하여 위치(인덱스)를 사용하여 직접 수행할 수 있습니다.

Creating

Creating a List

```
# List of integers
a = [1, 2, 3, 4, 5]

# List of strings
b = ['apple', 'banana', 'cherry']

# Mixed data types
c = [1, 'hello', 3.14, True]

print(a)
print(b)
print(c)
```

```
# From a tuple(string, tuple, or another list)
a = list((1, 2, 3, 'apple', 4.5))

print(a)
```

We can also create a list by passing an iterable (like a string, tuple, or another list) to list() function.

```
# Create a list [2, 2, 2, 2, 2]
a = [2] * 5

# Create a list [0, 0, 0, 0, 0, 0, 0]
b = [0] * 7

print(a)
print(b)
```

We can create a list with repeated elements using the multiplication operator.

A Python list is a collection that is ordered and changeable. In Python, lists are written with square brackets.

Accessing

Accessing List Elements

Elements in a list can be accessed using indexing. Python indexes start at 0, so a[0] will access the first element, while negative indexing allows us to access elements from the end of the list. Like index -1 represents the last elements of list.

```
a = [10, 20, 30, 40, 50]

# Access first element
print(a[0])

# Access last element
print(a[-1])
```

Adding

Adding Elements into List

We can add elements to a list using the following methods:

append(): Adds an element at the end of the list.
extend(): Adds multiple elements to the end of the list.
insert(): Adds an element at a specific position.

```
# Initialize an empty list
a = []

# Adding 10 to end of list
a.append(10)
print("After append(10):", a)

# Inserting 5 at index 0
a.insert(0, 5)
print("After insert(0, 5):", a)
```

```
# Adding multiple elements  [15, 20, 25] at the end
a.extend([15, 20, 25])
print("After extend([15, 20, 25]):", a)
```

Removing

Removing Elements from List

We can remove elements from a list using:

remove(): Removes the first occurrence of an element.
pop(): Removes the element at a specific index or the last element if no index is specified.
del statement: Deletes an element at a specified index.

```
a = [10, 20, 30, 40, 50]

# Removes the first occurrence of 30
a.remove(30)
print("After remove(30):", a)

# Removes the element at index 1 (20)
popped_val = a.pop(1)
print("Popped element:", popped_val)
print("After pop(1):", a)
```

```
# Deletes the first element (10)
del a[0]
print("After del a[0]:", a)
```

Iterating

Iterating Over Lists

We can iterate the Lists easily by using a for loopor other iteration methods. Iterating over lists is useful when we want to do some operation on each item or access specific items based on certain conditions. Let's take an example to iterate over the list using for loop.

```
a = ['apple', 'banana', 'cherry']

# Iterating over the list
for item in a:
    print(item)
```

Nested Lists

Nested Lists in Python

A nested list is a list within another list, which is useful for representing matrices or tables. We can access nested elements by chaining indexes.

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Access element at row 2, column 3
print(matrix[1][2])
```

Length

How To Find the Length of a List in Python

The length of a list refers to the number of elements in the list. There are several methods to determine the length of a list in Python. In this article, we'll explore different methods to find the length of a list. The simplest and most common way to do this is by using the len() function. Let's take an example.

Using len()

```
a = [1, 2, 3, 4, 5]
length = len(a)
print(length)
```

Using loop (Naive method)

```
a = [1, 2, 3, 4, 5]

# Initialize a counter to zero
count = 0

# Loop through each element in the list
for val in a:

    # Increment the counter for each element
    count += 1

print(count)
```

Length

Using length_hint()

The length_hint() function from the operator module provides an estimated length (not a guaranteed accurate count) of an iterable. This method is not typically used in regular code where len() is applicable.

```
from operator import length_hint

a = [1, 2, 3, 4, 5]
length = length_hint(a)
print(length)
```

Which method to choose?

len() Function: Best method, O(1) time complexity.
Naive Method: Useful for understanding iterations, but not recommended for practical use due to O(n) time complexity.
length_hint() from operator: Alternative with O(1) time complexity but len() would be preferred for finding length of list.

Find Largest Number

Python Program to Find Largest Number in a List

Finding the largest number in a list is a common task in Python. There are multiple way to do but the simplest way to find the largest in a list is by using Python's built-in max() function:

Using max()

Python provides a built-in max() function that returns the largest item in a list or any iterable. The time complexity of this approach is O(n) as it traverses through all elements of an iterable.

```
a = [10, 24, 76, 23, 12]

# Max() will return the largest in 'a'
largest = max(a)
print(largest)
```

Finding Largest Number

Using a Loop (Native method)

If we want to find the largest number in a list without using any built-in method (i.e. max()) function then can use a loop (for loop).

```
a = [10, 24, 76, 23, 12]

# Assuming first element is largest.
largest = a[0]

# Iterate through list and find largest
for val in a:
    if val > largest:

        # If current element is greater than largest
        # update it
        largest = val

print(largest)
```

Finding Largest Number

Using reduce() function from functools

Another method to find the largest number in a list is by using the reduce()function along with a lambda expression.

```
from functools import reduce

a = [10, 24, 76, 23, 12]

# Find the largest number using reduce
largest = reduce(lambda x, y: x if x > y else y, a)

print(largest)
```

Explanation: reduce(lambda x, y: x if x > y else y, a): The reduce() function takes pairs of elements in the list a and applies the lambda function to return the largest value. The lambda function compares x and y and returns the larger of the two.

Finding Largest Number

Using sort()

The sort() method is one of the quick method to get the largest value from list but this approach has a higher time complexity (O(n log n)) compared to using max() which is O(n) time.

```
a = [10, 24, 76, 23, 12]

a.sort()
largest = a[-1]

print(largest)
```

Which method to choose?

Using max(): This is easiest and most recommended way.
Using a Loop: This method is good if we want to solve manually.
Using reduce(): An advanced option for those interested in functional programming.
Using sort(): This method is not recommended here due to its higher time complexity.

Swap

Python Program to Swap Two Elements in a List

In this article, we will explore various methods to swap two elements in a list in Python. The simplest way to do is by using multiple assignment.

Using multiple assignment

```
a = [10, 20, 30, 40, 50]

# Swapping elements at index 0 and 4
# using multiple assignment
a[0], a[4] = a[4], a[0]

print(a)
```

Using comma operator

```
a = [10, 20, 30, 40, 50]

# Using a temporary variable
# to swap elements at index 2 and 4
temp = a[2]
a[2] = a[4]
a[4] = temp

print(a)
```

Swap

Using XOR

```
# Python program to demonstrate
# Swapping of two variables

x = 10
y = 50

# Swapping using xor
x = x ^ y
y = x ^ y
x = x ^ y

print("Value of x:", x)
print("Value of y:", y)
```

Using addition and subtraction operator

```
# Python program to demonstrate
# swapping of two variables

x = 10
y = 50

# Swapping of two variables
# using arithmetic operations
x = x + y
y = x - y
x = x - y

print("Value of x:", x)
print("Value of y:", y)
```

Swap

Using multiplication and division operator

```
# Python program to demonstrate
# swapping of two variables

x = 10
y = 50

# Swapping of two numbers
# Using multiplication operator

x = x * y
y = x / y
x = x / y

print("Value of x : ", x)
print("Value of y : ", y)
```

Using multiplication and division operator

```
#Python program to demonstrate
#swapping of two numbers
a = 5
b = 1
a = (a & b) + (a | b)
b = a + (~b) + 1
a = a + (~b) + 1
print("a after swapping: ", a)
print("b after swapping: ", b)
```

Checking element

Check if element exists in list in Python

In this article, we will explore various methods to check if element exists in list in Python. The simplest way to check for the presence of an element in a list is using the in Keyword.

Using in keyword

```
a = [10, 20, 30, 40, 50]

# Check if 30 exists in the list
if 30 in a:
    print("Element exists in the list")
else:
    print("Element does not exist")
```

Using a loop

```
a = [10, 20, 30, 40, 50]

# Check if 30 exists in the list using a loop
key = 30
flag = False

for val in a:
    if val == key:
        flag = True
        break

if flag:
    print("Element exists in the list")
else:
    print("Element does not exist")
```

Checking element

Using any()

The any() function is used to check if any element in an iterable evaluates to True. It returns True if at least one element in the iterable is truthy (i.e., evaluates to True), otherwise it returns False

```
a = [10, 20, 30, 40, 50]

# Check if 30 exists using any() function
flag = any(x == 30 for x in a)

if flag:
    print("Element exists in the list")
else:
    print("Element does not exist")
```

Checking element

Using count()

The count() function can also be used to check if an element exists by counting the occurrences of the element in the list. It is useful if we need to know the number of times an element appears.

```
a = [10, 20, 30, 40, 50]

# Check if 30 exists in the list using count()
if a.count(30) > 0:
    print("Element exists in the list")
else:
    print("Element does not exist")
```

Checking element

Which method to choose?

in Statement: The simplest and most efficient method for checking if an element exists in a list. Ideal for most cases.
for Loop: Allows manual iteration and checking and provides more control but is less efficient than using ‘in’.
any(): A concise option that works well when checking multiple conditions or performing comparisons directly on list.
count(): Useful for finding how many times an element appears in the list but less efficient for checking existance of element only.

Find index

Python List index() – Find Index of Item

List index() method searches for a given element from the start of the list and returns the position of the first occurrence.
Using index() method in Python, you can find position of the first occurrence of an element in list.

Using List index()

```
# list of animals
Animals= ["cat", "dog", "tiger"]
# searching positiion of dog
print(Animals.index("dog"))

#List of fruits
fruits = ["apple", "banana","cherry","apple"]
#Searching index of apple
print(fruits.index("apple"))
```

Find index

Example 1: Working on the index() With Start and End Parameters

In this example, we find an element in list python, the index of an element of 4 in between the index at the 4th position and ending with the 8th position.

```
# list of items
list1 = [1, 2, 3, 4, 1, 1, 1, 4, 5]

# Will print index of '4' in sublist
# having index from 4 to 8.
print(list1.index(4, 4, 8))
```

Find index

Example 2: Working of the index() With two Parameters only

In this example, we will see when we pass two arguments in the index function, the first argument is treated as the element to be searched and the second argument is the index from where the searching begins.

```
# list of items
list1 = [6, 8, 5, 6, 1, 2]

# Will print index of '6' in sublist
# having index from 1 to end of the list.
print(list1.index(6, 1))
```

Find index

Example 3: Index of the Element not Present in the List

Python List index() raises ValueError when the search element is not present inside the List.

```
# Python3 program for demonstration
# of index() method error

list1 = [1, 2, 3, 4, 1, 1, 1, 4, 5]

# Return ValueError
print(list1.index(10))
```

Find index

Example 4: How to fix list index out of range using Index()

Here we are going to create a list and then try to iterate the list using the constant values in for loops.

```
li = [1,2 ,3, 4, 5]

for i in range(6):
    print(li[i])
```

Reason for the error: The length of the list is 5 and if we are an iterating list on 6 then it will generate the error.

Solving this error without using len():
To solve this error we will take the count of the total number of elements inside the list and run a loop after that in the range of that count.

```
li = [1, 5, 3, 2, 4]
count=0

for num in li:
  count+=1
```

```
for i in range(count):
    print(li[i])
```

List to set

Ways to remove duplicates from list in Python

In this article, we'll learn several ways to remove duplicates from a list in Python. The simplest way to remove duplicates is by converting a list to a set.

Using set()

We can use set()to remove duplicates from the list. However, this approach does not preserve the original order.

```
a = [1, 2, 2, 3, 4, 4, 5]

# Remove duplicates by converting to a set
a = list(set(a))

print(a)
```

List to set

Using for loop

To remove duplicates while keeping the original order, we can use a loop (for loop) to add only unique items to a new list.

```
a = [1, 2, 2, 3, 4, 4, 5]

# Create an empty list to store unique values
res = []

# Iterate through each value in the list 'a'
for val in a:

    # Check if the value is not already in 'res'
    if val not in res:

        # If not present, append it to 'res'
        res.append(val)

print(res)
```

List to set

Using List Comprehension

List comprehension is another way to remove duplicates while preserving order. This method provides a cleaner and one-liner approach.

```
a = [1, 2, 2, 3, 4, 4, 5]

# Create an empty list to store unique values
res = []

# Use list comprehension to append values to 'res'
# if they are not already present
[res.append(val) for val in a if val not in res]

print(res)
```

List to set

Using Dictionary fromkeys()

Dictionaries maintain the order of items and by using dict.fromkeys() we can create a dictionary with unique elements and then convert it back to a list.

```
a = [1, 2, 2, 3, 4, 4, 5]

# Remove duplicates using dictionary fromkeys()
a = list(dict.fromkeys(a))

print(a)
```

Reversing

Reversing a List in Python

In this article, we are going to explore multiple ways to reverse a list. Python provides several methods to reverse a list using built-in functions and manual approaches. The simplest way to reverse a list is by using the reverse() method.
Let's take an example to reverse a list using reverse() method.

The reverse() method reverses the elements of the list in-place and it modify the original list without creating a new list. This method is efficient because it doesn't create a new list.

Using reverse()

```
a = [1, 2, 3, 4, 5]

# Reverse the list in-place
a.reverse()
print(a)
```

Reversing

Using List Slicing

Another efficient way to reverse a list is by using list slicing. This method creates a new list that contains the elements of the original list in reverse order.

```
a = [1, 2, 3, 4, 5]

# Create a new list that is a reversed list
# of 'a' using slicing
rev = a[::-1]

print(rev)
```

Reversing

Using the reversed()

Python's built-in reversed() function is another way to reverse the list. However, reversed() returns an iterator, so it needs to be converted back into a list.
The reversed() function returns an iterator, which is then converted into a list using the list() constructor.
Unlike reverse(), this does not modify the original list.

```
a = [1, 2, 3, 4, 5]

# Use reversed() to create an iterator
# and convert it back to a list
rev = list(reversed(a))
print(rev)
```

Reversing

Using a Loop (creating new reversed list)

If we want to reverse a list manually, we can use a loop (for loop) to build a new reversed list. This method is less efficient due to repeated insertions and extra space required to store the reversed list.

```
a = [1, 2, 3, 4, 5]

# Initialize an empty list to store reversed element
res = []

# Loop through each item and insert
# it at the beginning of new list
for val in a:
    res.insert(0, val)
print(res)
```

Reversing

Using a Loop (In-place reverse)

This method is an optimized version of the above method. We can use this method if we want to reverse the list in-place (without creating new list) using for loop.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Define the start and end indices
start, end = 2, 6

# Reverse the elements from index 2 to 6
while start < end:

    # Swap elements at start and end
    a[start], a[end] = a[end], a[start]

    # Move the start index forward
    start += 1

    # Move the end index backward
    end -= 1

print(a)
```

Reversing

Using List Comprehension

We can also use list comprehension to reverse the list, although this is less common than the other methods mentioned.

```
a = [1, 2, 3, 4, 5]

# Use list comprehension to create
# a reversed version of the list
rev = [a[i] for i in range(len(a) - 1, -1, -1)]
print(rev)
```

Reversing

Which method to choose?

reverse(): Use for in-place modification when we don't need the original list
List Slicing ([::-1]): Use to quickly create a reversed copy without modifying the original list
reversed(): Ideal for creating an iterator to reverse without modifying the original list and if we need an iterable for further operations.
Loop (In-Place): Use for more control during in-place reversal and especially if additional conditions are involved.

Avoid using list comprehensions or loops that create new lists because they are less efficient and more complex.The reverse() method or list slicing ([::-1]) are generally recommended for their simplicity and efficiency.

Sum

Python program to find sum of elements in list

In this article, we will explore various method to find sum of elements in list. The simplest and quickest way to do this is by using the sum() function.

Using sum()

```
a = [10, 20, 30, 40]
res = sum(a)
print(res)
```

Explanation: The sum() function takes an iterable as its argument and returns the sum of its elements.

Sum

Using a loop

```
a = [10, 20, 30, 40]

# Initialize a variable to hold the sum
res = 0

# Loop through each value in the list
for val in a:

    # Add the current value to the sum
    res += val

print(res)
```

To calculate the sum without using any built-in method, we can use a loop (for loop).

Sum

Frequently Asked Question on Finding Sum of Elements in List

What is sum() function in Python?

The sum() function is a built-in Python function that takes an iterable (such as a list) and returns the sum of its elements.

Can I use sum() function for non-numeric elements?

No, sum() function works with numeric data types. If you try to use it with non-numeric elements, you will get a TypeError.

Can I sum elements in a nested list?

To sum elements in a nested list, you need to flatten the list before applying the sum() function. This can be done using loops or using itertools.chain(), which help to store all elements into a single list.

What happens if my list is empty?

If your list is empty, using sum() function will return 0 by default. This is because there are no elements to add and the default return value of the function is 0.

Merge

Merge Two Lists in Python

Python provides several approaches to merge two lists. In this article, we will explore different methods to merge lists with their use cases. The simplest way to merge two lists is by using the + operator.

Using + operator

```
a = [1, 2, 3]
b = [4, 5, 6]

# Merge the two lists and assign
# the result to a new list
c = a + b
print(c)
```

Explanation: The + operator creates a new list by concatenating a and b. This operation does not modify the original lists but returns a new combined list.

Merge

Using extand()

Another common method is to use the extend() function, which modifies the original list by adding elements from another list.

```
a = [1, 2, 3]
b = [4, 5, 6]

# Add all elements from list 'b' to the end of list 'a'
a.extend(b)

print(a)
```

Explanation: The extend() is an in-place operation that appends each element of list b to list a. This means that the original list a gets updated without creating a new list, which makes it more memory efficient for large lists.

Merge

Using the * Operator

We can use the * operator to unpack the elements of multiple lists and combine them into a new list.

```
a = [1, 2, 3]
b = [4, 5, 6]

# Use the unpacking operator to merge the lists
c = [*a, *b]

print(c)
```

Explanation: The * operator unpacks the elements of a and b, placing them directly into the new list c.

Merge

Using for loop

We can also merge two lists using a simple for loop. This approach is provides more control if we need to perform additional operations on the elements while merging.

```
a = [1, 2, 3]
b = [4, 5, 6]

# Initialize an empty list to store the merged elements
res = []

# Append all elements from the first list
for val in a:
    res.append(val)

# Append all elements from the second list
for val in b:
    res.append(val)

print(res)
```

Merge

Using list comprehension

List comprehension is an efficient and concise alternative to the for loop for merging lists. It allows us to iterate through each list and merge them into a new one in a single line of code.

```
a = [1, 2, 3]
b = [4, 5, 6]

# Use list comprehension to create a new merged list
c = [item for item in a] + [item for item in b]

print(c)
```

Explanation: List comprehension iterates over both lists a and b, creating a new list by adding all items from each list.

Merge

Using the itertools.chain()

The itertools.chain() method from the itertools module provides an efficient way to merge multiple lists. It doesn't create a new list but returns an iterator, which saves memory, especially with large lists.

```
# imports chain function from itertools module
from itertools import chain

a = [1, 2, 3]
b = [4, 5, 6]

# Use itertools.chain to merge the lists in an efficient manner
c = list(chain(a, b))

print(c)
```

Explanation: list(chain(a, b)) combines the lists a and b. The chain(a, b) creates an iterator that collect all items from a first then all items from b. Then, change this iterator into a list will gives us merged list.

Merge

Which method to choose?

+ Operator: Quick and simple for small lists but can consume a lot of memory for larger ones.
extend() Method: Efficiently merges large lists by modifying the original list in place.
Unpacking (*): It provides a clean syntax, however, it may not be optimal for very large lists due to memory usage.
itertools.chain(): It works best for large lists. It merges without creating additional memory.
For Loop: It provides more control to use complex merging conditions.
List Comprehension: This is an alternative approach of using "for loop", it has concise and cleaner syntax.

Sort

Sort a list in python

Sorting is a fundamental operation in programming, allowing you to arrange data in a specific order. Here is a code snippet to give you an idea about sorting.

Using sort()

```
# Initializing a list
a = [5, 1, 5, 6]

# Sort modifies the given list
a.sort()
print(a)

b = [5, 2, 9, 6]

# Sorted does not modify the given list
# and returns a different sorted list
bs = sorted(b)
print(b)
print(bs)
```

Sort

Python sort() Method

The method sorts the list in place and modifies the original list. By default, it sorts the list in ascending order.

```
# Initializing a list
a = [5, 2, 9, 1, 5, 6]

# Sorting the list in ascending order
a.sort()
print("Sorted list (ascending):", a)

a.sort(reverse=True)
print("Sorted list (descending):", a)
```

Sort

Python sorted() Method

Python sorted() function returns a sorted list. It is not only defined for the list and it accepts any iterable (list, tuple, string, etc.). It does not modify the given container and returns a sorted version of it.

```
# Initializing a list
a = [5, 2, 9, 1, 5, 6]

# Sorting the list in descending order
sa = sorted(a)
print("Sorted list (ascending):", sa)

sa = sorted(a, reverse=True)
print("Sorted list (descending):", sa)
```

Sort

Sorting Other Containers :

Sorting a tuple: The function converts the tuple into a sorted list of its elements.
Sorting a set: Since sets are unordered, converts the set into a sorted list of its elements.
Sorting a string: This will sort the characters in the string and return a list of the sorted characters.
Sorting a dictionary: When a dictionary is passed to , it sorts the dictionary by its keys and returns a list of the keys in sorted order.
Sorting a list of tuples: The list of tuples is sorted primarily by the first element of each tuple, and secondarily by the second element if the first ones are identical.

```
# Sorting a tuple
a = (10, 12, 5, 1)
print(sorted(a))

# Sorting a set
s = {'gfg', 'course', 'python'}
print(sorted(s))

# Sorting a string
st = 'gfg'
print(sorted(st))

# Attempting to sort a dictionary (it will sort the keys)
d = {10: 'gfg', 15: 'ide', 5: 'course'}
print(sorted(d))

# Sorting a list of tuples
l = [(10, 15), (1, 8), (2, 3)]
print(sorted(l))
```

Sort

Sorting User Defined Object

This code defines a class with an initializer that sets the and coordinates for point objects. The function takes a point object and returns its coordinate. A list of objects is created, and then sorted based on their values using the function as the key in the method. Finally, it prints out the and coordinates of each in the sorted list.

Example 1: Using a Separate Method

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def myFun(p):
    return p.x

l = [Point(1, 15), Point(10, 5), Point(3, 8)]
l.sort(key=myFun)

for i in l:
    print(i.x, i.y)
```

Sort

Example 2: Using a __lt__ Method

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __lt__(self, other):
        return self.x < other.x

l = [Point(1, 15), Point(10, 5), Point(3, 8)]
l.sort()

for i in l:
    print(i.x, i.y)
```

Sort

Sorting with Custom Criteria

Sorting Based on Absolute Values

Sometimes you may want to sort numbers based on their absolute values while preserving their original signs. You can do this by using the parameter in both the method and the function.

```
a = [1, -5, 10, 6, 3, -4, -9]

# Sorting by absolute values in descending order
sa = sorted(a, key=abs, reverse=True)
print("Sorted by absolute values:", sa)
```

Sort

Sorting with Custom Criteria

Custom Sorting with Lambda Functions

You can use lambda functions to define custom sorting logic. For example, if you want to sort a list of tuples based on the second element:

```
# List of tuples
a = [(1, 'one'), (3, 'three'), (2, 'two')]

# Sorting by the second element of each tuple
sa = sorted(a, key=lambda x: x[1])
print("Sorted by second element:", sa)
```

Performance Considerations

When sorting lists, the choice between and may depend on whether you want to maintain the original list. The method is generally faster since it sorts the list in place. However, if you need to keep the original order, use .

Check sublist

Python – Check for Sublist in List

Sometimes, while working with Python Lists, we can have a problem in which we need to check if a particular sublist is contained in an exact sequence in a list. This task can have application in many domains such as school programming and web development. Let's discuss certain ways in which this task can be performed.

Input : test_list = [5, 6, 3, 8, 2, 1, 7, 1], sublist = [8, 2, 3]
Output : False

Input : test_list = [5, 6, 3, 8, 2, 3, 7, 1], sublist = [8, 2, 3]
Output : True

Check sublist

Method #1 : Using loop + list slicing

The combination of the above functions can be used to solve this problem. In this, we perform the task of checking for sublist by incremental slicing using the list slicing technique.

```
# Python3 code to demonstrate working of
# Check for Sublist in List
# Using loop + list slicing

# initializing list
test_list = [5, 6, 3, 8, 2, 1, 7, 1]

# printing original list
print("The original list : " + str(test_list))

# initializing sublist
sublist = [8, 2, 1]
```

```
# Check for Sublist in List
# Using loop + list slicing
res = False
for idx in range(len(test_list) - len(sublist) + 1):
    if test_list[idx: idx + len(sublist)] == sublist:
        res = True
        break

# printing result
print("Is sublist present in list ? : " + str(res))
```

Check sublist

Method #2 : Using any() + list slicing + generator expression

The combination of the above functions is used to solve this problem. In this, we perform the task of checking for any sublist equating to desired using any(), and list slicing is used to slice incremental list of the desired length.

```
# Python3 code to demonstrate working of
# Check for Sublist in List
# Using any() + list slicing + generator expression

# initializing list
test_list = [5, 6, 3, 8, 2, 1, 7, 1]

# printing original list
print("The original list : " + str(test_list))

# initializing sublist
sublist = [8, 2, 1]

# Check for Sublist in List
# Using any() + list slicing + generator expression
res = any(test_list[idx: idx + len(sublist)] == sublist
          for idx in range(len(test_list) - len(sublist) + 1))

# printing result
print("Is sublist present in list ? : " + str(res))
```

Check sublist

Method #3 : Using in operator

In this, we will convert both the list to a string using join() method and map() method and then later use in operator to check for the sublist in list.

```
# Python3 code to demonstrate working of
# Check for Sublist in List
# Using in operator

# initializing list
test_list = [5, 6, 3, 8, 2, 1, 7, 1]

# printing original list
print("The original list : " + str(test_list))

# initializing sublist
sublist = [8, 2, 1]
```

```
# Check for Sublist in List
res = ''.join(map(str, sublist)) in ''.join(map(str, test_list))

# printing result
print("Is sublist present in list ? : " + str(res))
```

Check sublist

Method #4: Using the in operator with zip() and *

Step-by-step algorithm:

1. Create the main list of integers to search for the sublist.
2. Create the sublist to search for in the main list.
3. Use a list comprehension to create a list of all possible sublists of the main list with the same length as the sublist.
4. Use the zip function to group the elements of each sublist created in step 3.
5. Compare each group of elements created in step 4 to the elements of the sublist.
6. If a group of elements matches the elements of the sublist, return True. Otherwise, return False.

Check sublist

```
#This is a Python code file to check whether a sublist is present in a list
#Initializing list
test_list = [5, 6, 3, 8, 2, 1, 7, 1]

#Initializing sublist to be checked
sublist = [8, 2, 1]

# printing original list
print("The original list : " + str(test_list))
#Using list comprehension and zip function to check for sublist
res = any(sublist == list(x) for x in zip(*[test_list[i:] for i in range(len(sublist))]))

#Printing the result
print("Is sublist present in list ? : " + str(res))
#This code is contributed by Vinay Pinjala.
```