AnB_Prime

——>1<——

List Data Type

Topic 03 Key concept

1. List

1.1 Efficient data processing

One of the important purposes of
using a computer is to process data efficiently.

1. List

1.1 Efficient data processing

```
1 score0 = 87
2 score1 = 84
3 score2 = 95
4 score3 = 67
5 score4 = 88
6 score5 = 94
7 score6 = 63
8 print(score0, score3)
```

```
87 67
```

- Let's consider a case where we need to store and process test score data for multiple individuals.
- We would need multiple variables to store the test scores.

1. List

1.1 Efficient data processing

```
1 score0 = 87
2 score1 = 84
3 score2 = 95
4 score3 = 67
5 score4 = 88
6 score5 = 94
7 score6 = 63
8 print(score0, score3)
```

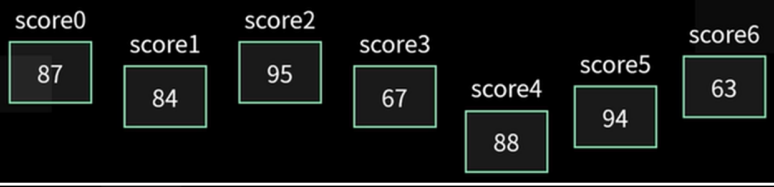
```
87 67
```

- In the following code, since there are 7 data points, we would need to define 7 individual variables and assign values to these variables.
- Manipulating and copying individual elements would require a lot of coding.

1. List

1.2 Importance of the list data type

- A memory structure consisting of the creation of individual variables and assignment of values would require many variable names.
- Performing operations such as sum and average on variables would result in complex coding and a higher chance of errors.
- It would be convenient to bundle them into a single data type.



1. List

1.3 What is a list?

- A list is a data structure that can hold individual values together.
  - It is more convenient and efficient than writing and managing multiple variable names.
  - It can be copied and manipulated all at once.

1. List

1.3 What is a list?

- Item or element
  - The individual data that make up the list is called item or element.
  - They are data values separated by commas when stored.
- The following is an example of defining a list by inserting items. Items in a list are separated by commas within square brackets [].

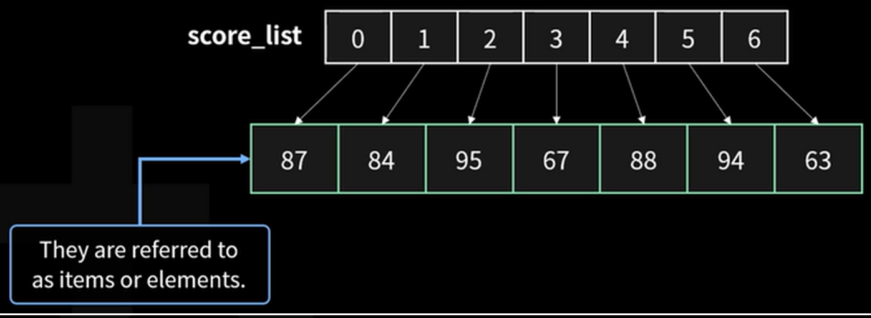
```
1 score_list = [87, 84, 95, 67, 88, 94, 63]
2 score_list
```

```
[87, 84, 95, 67, 88, 94, 63]
```

1. List

1.3 What is a list?

- The continuous data values for storing test scores for multiple individuals are stored in the list variable “score_list” as shown below.



1. List

1.3 What is a list?

Index	Indexing
• The number that points to the item value in the list is called the index.	• Referring to getting an item, it is called indexing. • It is the act of accessing data values using the index of the item.

1. List

1.3 What is a list?

- The following is an example of defining a list and referencing items.

```
1 score_list = [87, 84, 95, 67, 88, 94, 63]
2 print(score_list[0], score_list[3])
```

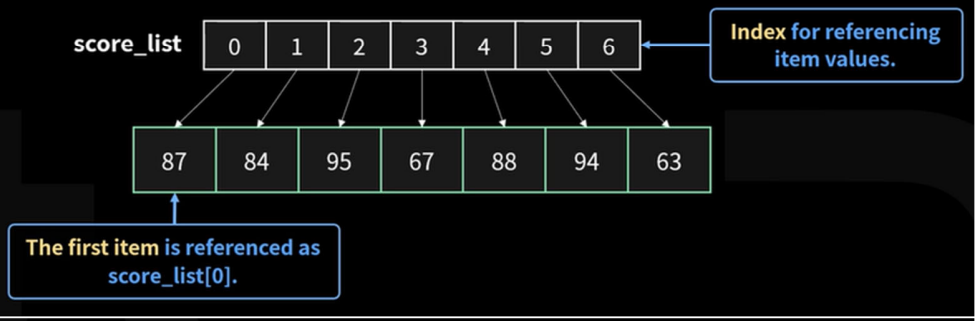
```
87 67
```

Item referencing uses the list name and index.

1. List

1.3 What is a list?

- Using an index, you can retrieve the desired value by specifying its position, such as the first value of the score_list variable.



2. Defining a list

2.1 Empty list

- In Python, lists can be created like other variables.
- That is, by separating the items within [ ] with commas and assigning them to a variable, a list variable is created.
- Let's create a list with 3 members of the idol group BTS as follows.

```
1 bts = ['V', 'Jungkook', 'Jimin']
```

- If there are no members initially, the following method is also possible. In this case, an empty list is created.

```
1 bts = []
```

2. Defining a list

2.1 Empty list

What is the use of an empty list with no items?

- In an empty list, items can be added through code.
- In many cases, we cannot predict how many items will be in the list. In such cases, an empty list is useful.

```
1 bts = []
2 bts.append("V")
3 bts
```

To add items to a list, the append(item) method is used.

```
['V']
```

2. Defining a list

2.1 Empty list

- Another simple way to add items to a list is by using the + operator.

```
1 bts = []
2 bts = bts + ["V"]
3 bts
```

```
['V']
```

2. Defining a list

2.2 Data types in a list

- There are no limitations on the data types that can be included in a list.
- In the mixed_list, both integer values like 100, 200, 400 and a string value ‘apple’ can be stored in the list simultaneously.

```
1 mixed_list = [100, 200, 'apple', 400]
2 print(mixed_list)
```

Python lists can have different data types such as integers, floats, and strings.

```
[100, 200, 'apple', 400]
```

2. Defining a list

2.3 Creating a list using the range function to generate a sequence of numbers

- A list can also be created using the range function.
- By using the function range(1, 10), we can obtain a sequence of numbers from 1 to 9 and then create a list with this sequence as elements using the list function.

```
1 list4 = list(range(1, 10))
2 list4
```

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

The range function in Python allows for quick generation of a sequence of consecutive numbers.

3. List indexing

3.1 Indexing and length of a list

index	[0]	[1]	[2]	[3]	[4]	[5]
n_list =	11	22	33	44	55	66

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3 print(len(n_list))
```

```
[11, 22, 33, 44, 55, 66]
6
```

The length of 'n_list' is 6.

- In a list, the position of an item starts with an index of 0 for the first item and goes up to n-1 for the last item.
- By using the len function, we can determine the number of items or the length of the list.

3. List indexing

3.1 Indexing and length of a list

- The accessible range in indexing is from 0 to 5.

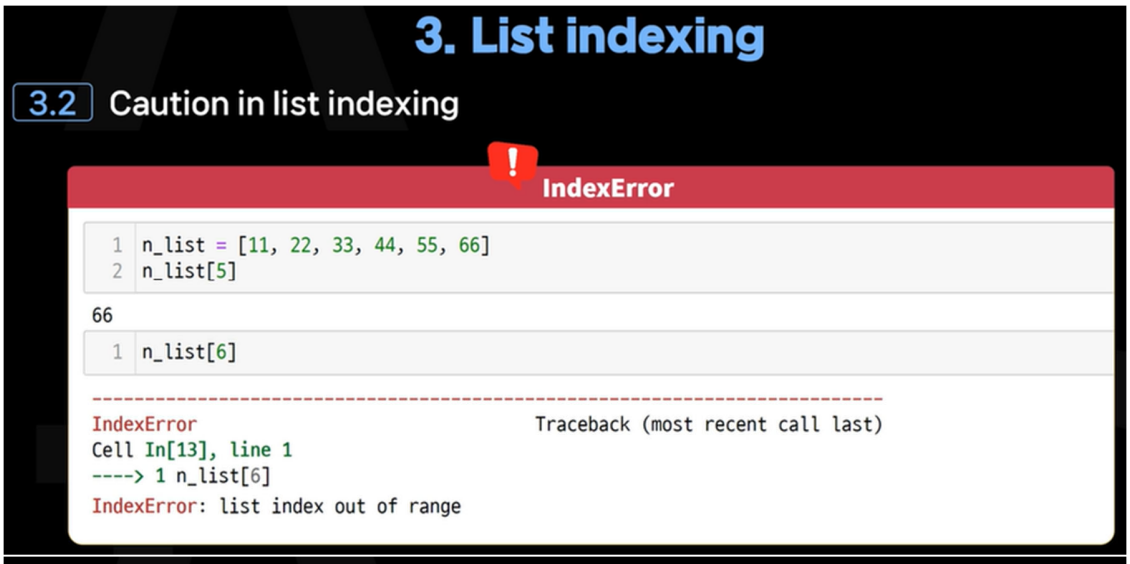
```
1 n_list[0]
```

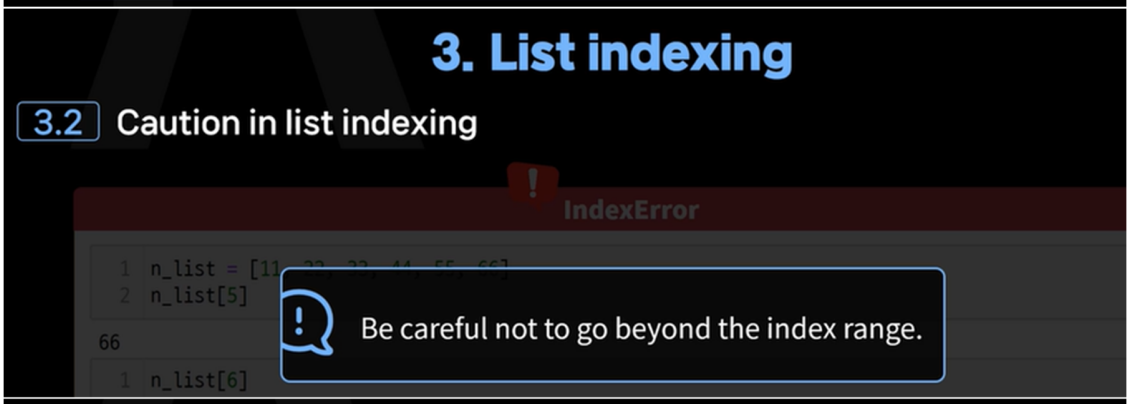
```
11
```

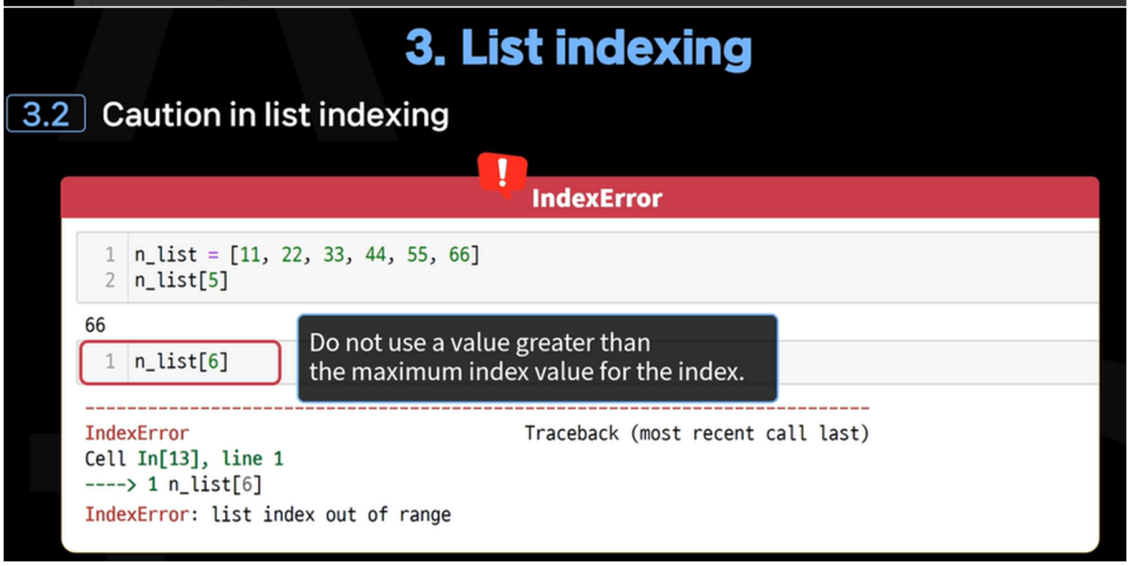
```
1 n_list[1]
```

```
22
```

n_list[0] = 11
n_list[1] = 22
n_list[2] = 33
n_list[3] = 44
n_list[4] = 55
n_list[5] = 66


3. List indexing
3.2 Caution in list indexing
IndexError
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[5]
66
1 n_list[6]
IndexError Traceback (most recent call last)
Cell In[13], line 1
----> 1 n_list[6]
IndexError: list index out of range


3. List indexing
3.2 Caution in list indexing
IndexError
Be careful not to go beyond the index range.


3. List indexing
3.2 Caution in list indexing
IndexError
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[5]
66
1 n_list[6]
Do not use a value greater than
the maximum index value for the index.
IndexError Traceback (most recent call last)
Cell In[13], line 1
----> 1 n_list[6]
IndexError: list index out of range

3. List indexing

3.2 Caution in list indexing

IndexError

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[5]
```

The maximum index is len(n_list)-1.

```
66
```

```
1 n_list[6]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 n_list[6]
IndexError: list index out of range
```

3. List indexing

3.2 Caution in list indexing

IndexError

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[5]
```

```
66
```

```
1 n_list[6]
```

If the index exceeds the accessible range, the following error occurs:
IndexError: list index out of range.

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 n_list[6]
IndexError: list index out of range
```

3. List indexing

3.3 Negative index

- It is also possible to access items using negative indexing.

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[-1]
```

```
66
```

```
1 n_list[-2]
```

```
55
```

```
1 n_list[-3]
```

```
44
```

3. List indexing

3.3 Negative index

- It is also possible to access items using negative indexing.

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 n_list[-1]
```

```
66
```

Line 2
- The last item value of a list can be accessed using the negative index -1.

```
1 n_list[-2]
```

```
55
```

```
1 n_list[-3]
```

```
44
```

3. List indexing

3.3 Negative index

- Python lists allow referencing elements using negative index.
- In the case of negative indexes, indexing starts from the last element with -1, -2, -3, and so on.

```
n_list[-1] = 66
n_list[-2] = 55
n_list[-3] = 44
n_list[-4] = 33
n_list[-5] = 23
n_list[-6] = 11
```

n_list =	11	22	33	44	55	66
negative index	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

3. List indexing

3.4 Glossary of Python terms: list, index, indexing

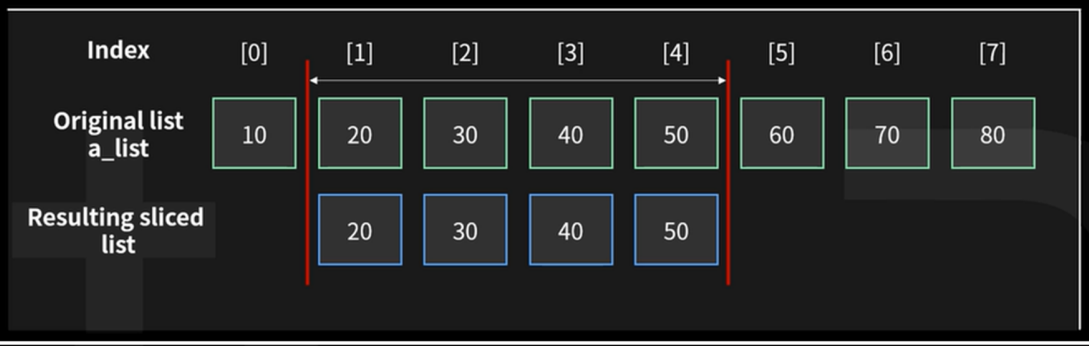
- Let's summarize the terms we have discussed so far.

List	- A Python data structure that can contain multiple items and allows for replacing internal items. - It also supports extracting the desired values from the stored items.
Index	- A number that points to the item value in the list. - The index of a list with n items increases from 0 to n-1. - Negative index can also be used.
Indexing	- The act of accessing data values by using the index of the item.

4. List slicing

4.1 Slicing

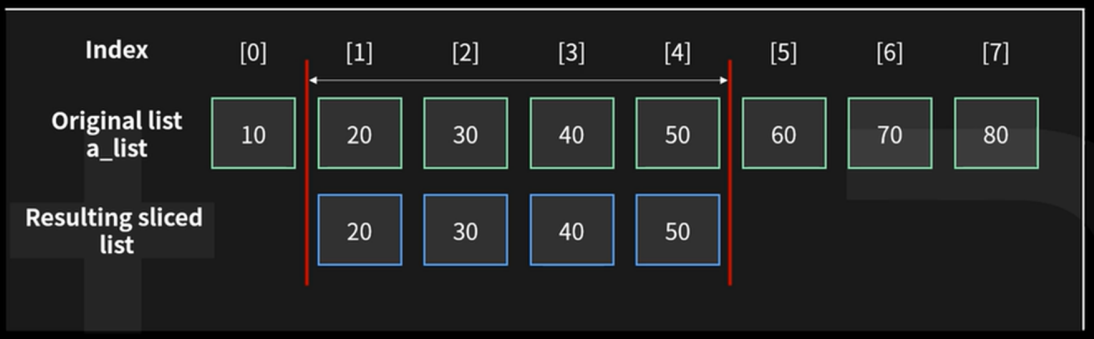
- Slicing: Refers to selecting specific segments of items within a list.



4. List slicing

4.1 Slicing

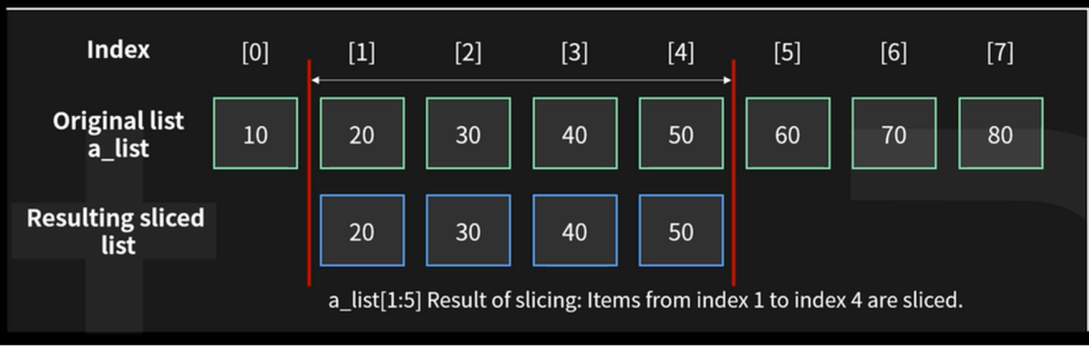
- To specify a range, we use the colon (:) and the syntax is list_name [start : end].
- You can slice the items up to the index minus 1 from the end value of the range.



4. List slicing

4.1 Slicing

- Slicing a list does not modify the original list; instead, a new list is created and returned.
- In other words, a slice is a partial copy of the original list.



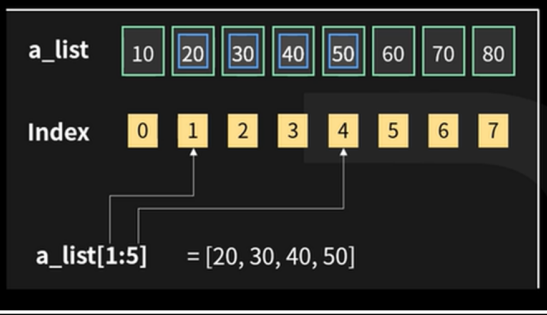
4. List slicing

4.2 Slicing Example: When both the starting and ending indexes are specified

- In Python, you can slice a list as follows:
  a_list[1:5]: Returns items from a_list[1] to a_list[5-1].

```
1 a_list = [10, 20, 30, 40, 50, 60, 70, 80]
2 a_list[1 : 5]
[20, 30, 40, 50]
```

Line 2
- Slicing from the second element, 20, to the fifth element, 50.
- The result is 20, 30, 40, 50.



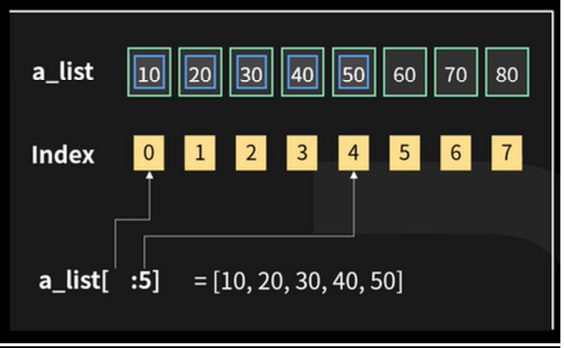
4. List slicing

4.3 Slicing Example: When the starting index is omitted

- a_list[:i]: Returns items from the beginning of the list, a_list[0], to a_list[i-1].

```
1 a_list[ : 5]
[10, 20, 30, 40, 50]
```

Line 1
- Slicing from the first element, 10, to the fifth element, 50.
- The result is 10, 20, 30, 40, 50.



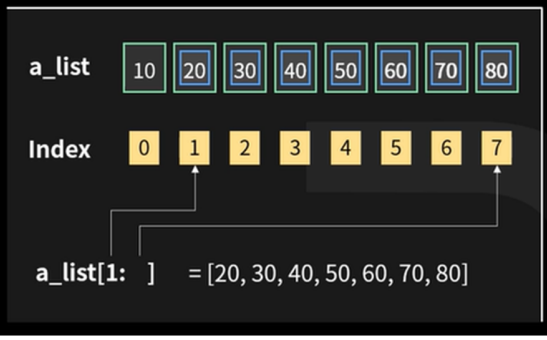
4. List slicing

4.4 Slicing Example: When the ending index is omitted

- a_list[i:]: Returns all items from a_list[i] to the last item in the list.
- The second index can be omitted when slicing up to the last element.

```
1 a_list[1 : ]
[20, 30, 40, 50, 60, 70, 80]
```

Line 1
- Slicing from the second element, 20, to the last element.



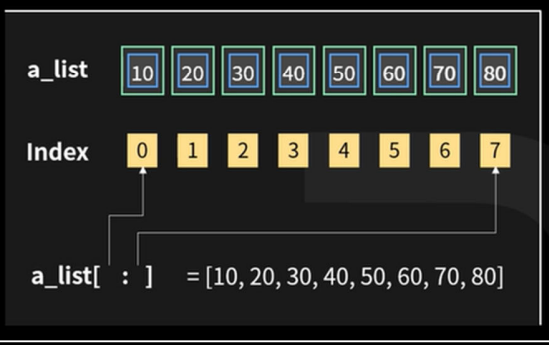
4. List slicing

4.5 Slicing Example: When both the starting and ending indexes are omitted

- [:] slicing: Returns all items in the list.

```
1 a_list[ : ]
[10, 20, 30, 40, 50, 60, 70, 80]
```

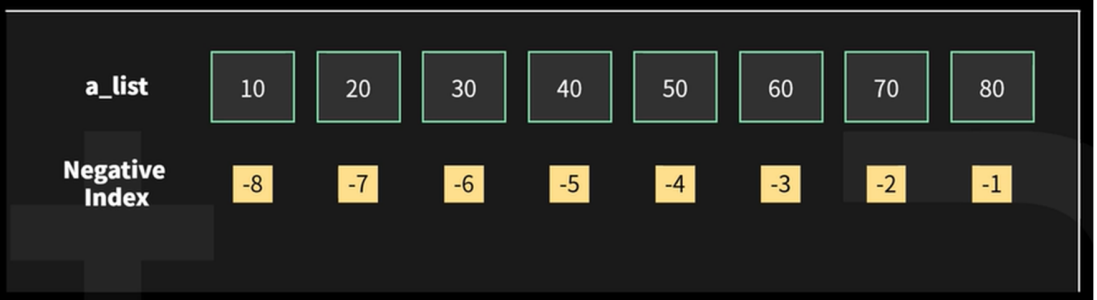
Line 1
- When both indexes are omitted, slicing includes all elements from the first to the last.



4. List slicing

4.6 Slicing Example: Slicing a list with negative index

- The index of the last element becomes -1, and the preceding elements are assigned -2, -3, and so on.

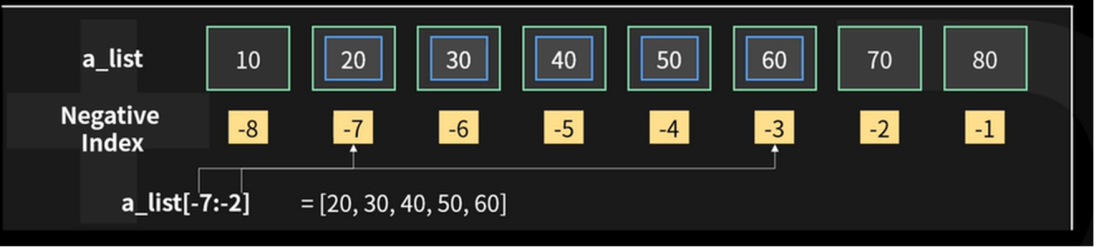


4. List slicing

4.6 Slicing Example: Slicing a list with negative index

- a_list[-7:-2]: Using the range of negative indexes, it returns items from a_list[-7] to a_list[-2-1]. Therefore, it includes the items [20, 30, 40, 50, 60].

```
1 a_list[-7 : -2]
[20, 30, 40, 50, 60]
```



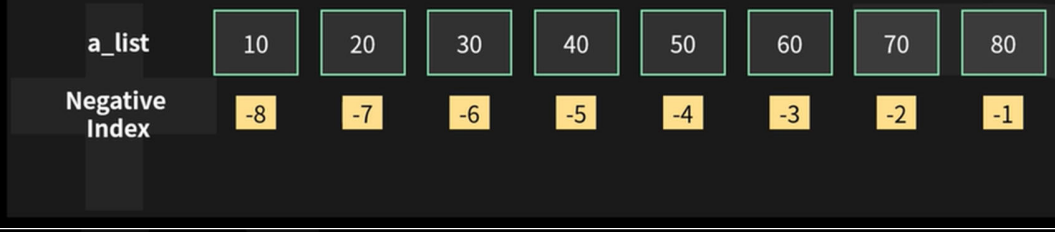
4. List slicing

4.7 Slicing Example: Omitting the last index in negative index

- When omitting the last index in negative indexing, it returns all items up to the last item in the list.

```
1 a_list[-7 : ]
```

```
[20, 30, 40, 50, 60, 70, 80]
```



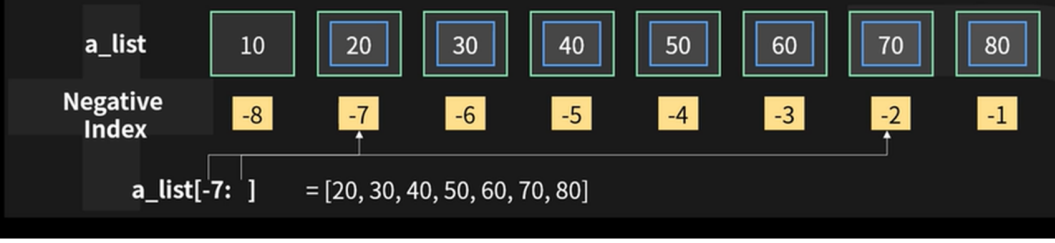
4. List slicing

4.7 Slicing Example: Omitting the last index in negative index

- When omitting the last index in negative indexing, it returns all items up to the last item in the list.

```
1 a_list[-7 : ]
```

```
[20, 30, 40, 50, 60, 70, 80]
```



4. List slicing

4.8 Slicing Example: Omitting the first index in negative index

- In slicing with negative indexes, it is also possible to omit the first index.
- a_list[:-2]: Returns items from the first item to the item at index (-2-1) = -3.

```
1 a_list[ : -2]
```

```
[10, 20, 30, 40, 50, 60]
```

4. List slicing

4.8 Slicing Example: Omitting the first index in negative index

- In slicing with negative indexes, it is also possible to omit the first index.
- a_list[:-2]: Returns items from the first item to the item at index (-2-1) = -3.

```
1 a_list[ : -2]
```

```
[10, 20, 30, 40, 50, 60]
```

4. List slicing

4.9 Slicing syntax

- Python's slicing supports negative index and negative step values.

Syntax	What it does
a_list[start:end]	Slices items from start to (end-1) (end index item is not included)
a_list[start:]	Slices items from start to the end of the list, i.e., the remaining part
a_list[:end]	Slices items from the beginning to index end-1
a_list[:]	Slices the entire list
a_list[start:end:step]	Slices items from start to end-1, skipping step items

4. List slicing

4.9 Slicing syntax

- Python's slicing supports negative index and negative step values.

Syntax	What it does
a_list[-2:]	Slices the last two items
a_list[:-2]	Slices all items except the last two
a_list[::-1]	Slices all items in reverse order
a_list[1::-1]	Slices items from index 1 to the end in reverse order

5. Lists and operators

5.1 Concatenating lists using the + operator

- The + operator can be used to concatenate two lists.
- In the following code, all items from person1 and person2 will be included in a single list.

```
1 person1 = ['David Doe', 20, 1, 180.0, 100.0]
2 person2 = ['John Smith', 25, 1, 170.0, 70.0]
3
4 person_list = person1 + person2
5 print(person_list)
```

```
['David Doe', 20, 1, 180.0, 100.0, 'John Smith', 25, 1, 170.0, 70.0]
```

5. Lists and operators

5.2 The 'in' Operator for Checking Values in a List

```
1 a_list = [10, 20, 30, 40]
2 10 in a_list
```

```
True
```

```
1 50 in a_list
```

```
False
```

```
1 10 not in a_list
```

```
False
```

```
1 50 not in a_list
```

```
True
```

- The membership operator **'in'** returns True or False, indicating whether an element is present or not in a specific sequence.
- It returns True if the element is present and False if it is not. (The **'not in'** operator returns the opposite value.)

6. List methods

6.1 The append Method for Adding Items to a List

- It allows adding desired items to an existing list.
- A method is a function specific to the list itself and is called using a dot (.) notation.

```
1 a_list = ['a', 'b', 'c', 'd', 'e']
2 a_list.append('f')
3 a_list
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

```
1 n_list = [10, 20, 30, 40]
2 n_list.append(50)
3 n_list
```

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

6. List methods

6.1 The append Method for Adding Items to a List

- It allows adding desired items to an existing list.
- A method is a function specific to the list itself and is called using a dot (.) notation.

```
1 a_list = ['a', 'b', 'c', 'd', 'e']
2 a_list.append('f')
3 a_list
```

append method
- Inserts a new item at the end of an existing list.

```
['a', 'b', 'c', 'd', 'e', 'f']
```

```
1 n_list = [10, 20, 30, 40]
2 n_list.append(50)
3 n_list
```

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

6. List methods

6.2 The extend Method for Adding Items to a List

```
1 list1 = ['a', 'b', 'c']
2 list2 = [1, 2, 3]
3 list1.extend(list2)
4 list1
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
1 list1.extend('d')
2 list1
```

```
['a', 'b', 'c', 1, 2, 3, 'd']
```

6. List methods

6.2 The extend Method for Adding Items to a List

```
1 list1 = ['a', 'b', 'c']
2 list2 = [1, 2, 3]
3 list1.extend(list2)
4 list1
```

```
['a', 'b', 'c', 1, 2, 3]
```

extend method
- This method adds a list or items at the end of a list.

```
1 list1.extend('d')
2 list1
```

```
['a', 'b', 'c', 1, 2, 3, 'd']
```

6. List methods

6.2 The extend Method for Adding Items to a List

```
1 list1 = ['a', 'b', 'c']
2 list2 = [1, 2, 3]
3 list1.extend(list2)
4 list1
```

```
['a', 'b', 'c', 1, 2, 3]
```

It can be used by passing a list as an argument.
In this case, it adds the elements of [1, 2, 3]
to the list [‘a’, ‘b’, ‘c’], resulting in [‘a’, ‘b’, ‘c’, 1, 2, 3].

```
1 list1.extend('d')
2 list1
```

```
['a', 'b', 'c', 1, 2, 3, 'd']
```

6. List methods

6.2 The extend Method for Adding Items to a List

```
1 list1 = ['a', 'b', 'c']
2 list2 = [1, 2, 3]
3 list1.extend(list2)
4 list1
```

```
['a', 'b', 'c', 1, 2, 3]
```

Individual elements like ‘d’ can also be inserted
as arguments.

```
1 list1.extend('d')
2 list1
```

```
['a', 'b', 'c', 1, 2, 3, 'd']
```

6. List methods

6.3 Difference between append and extend Functions

- If we try to append list1 as shown below,
  it will not give the expected result of [11, 22, 33, 44, 55, 66].
- When we call the list1.append([55, 66]) method,
  it adds the entire list [55, 66] as an item after [11, 22, 33, 44].

```
1 list1 = [11, 22, 33, 44]
2 list1.append([55, 66])
3 list1
```

```
[11, 22, 33, 44, [55, 66]]
```

Therefore, list1 becomes [11, 22, 33, 44, [55, 66]].

6. List methods

6.3 Difference between append and extend Functions

- In such cases, the extend method can be used to add new items to a list.

```
1 list1 = [11, 22, 33, 44]
2 list1.extend([55, 66])
3 list1
```

```
[11, 22, 33, 44, 55, 66]
```

6. List methods

6.4 The insert Method for Adding Items to a List

- While append can only add items at the end of a list, insert(index, item) method adds an item at the specified index.

```
1 slist = ['David', 178.9, 'John', 173.5, 'Jane', 176.1]
2 print(slist)
3 slist.insert(4, "Petter")
4 slist.insert(5,168.1)
5 print(slist)
```

```
['David', 178.9, 'John', 173.5, 'Jane', 176.1]
['David', 178.9, 'John', 173.5, 'Petter', 168.1, 'Jane', 176.1]
```

6. List methods

6.5 The remove Method for Deleting Items from a List

- It is possible to delete desired items from an existing list using the methods provided by the list.
- For example, we can use remove(44).

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 n_list.remove(44)
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

6. List methods

6.5 The remove Method for Deleting Items from a List

- It is possible to delete desired items from an existing list using the methods provided by the list.
- For example, we can use remove(44).

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 n_list.remove(44)
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

Line 4
- Using the remove method of the n_list list to delete the item with the value 44.

6. List methods

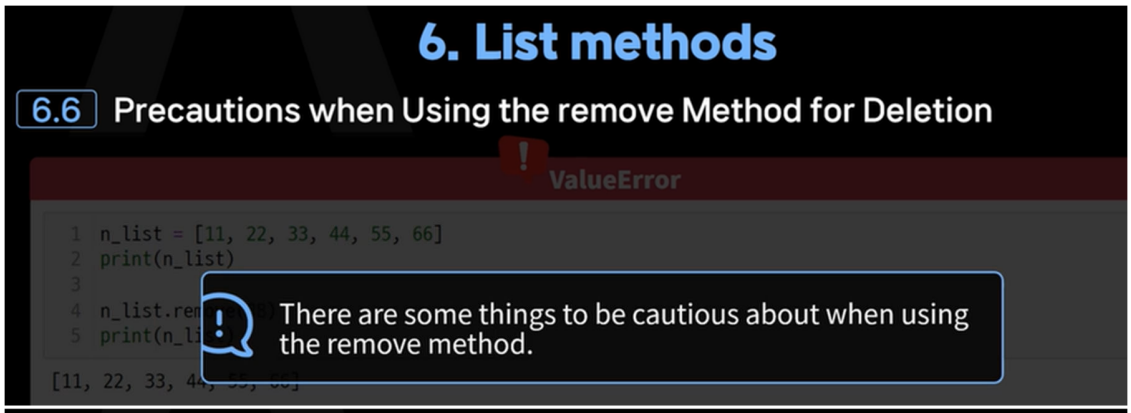
6.6 Precautions when Using the remove Method for Deletion

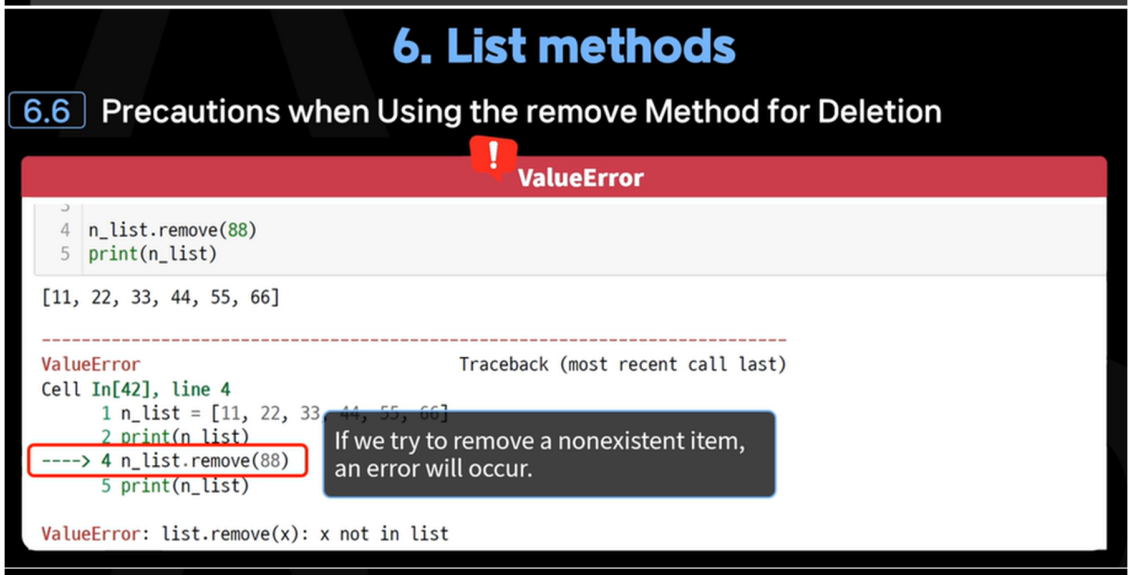
ValueError

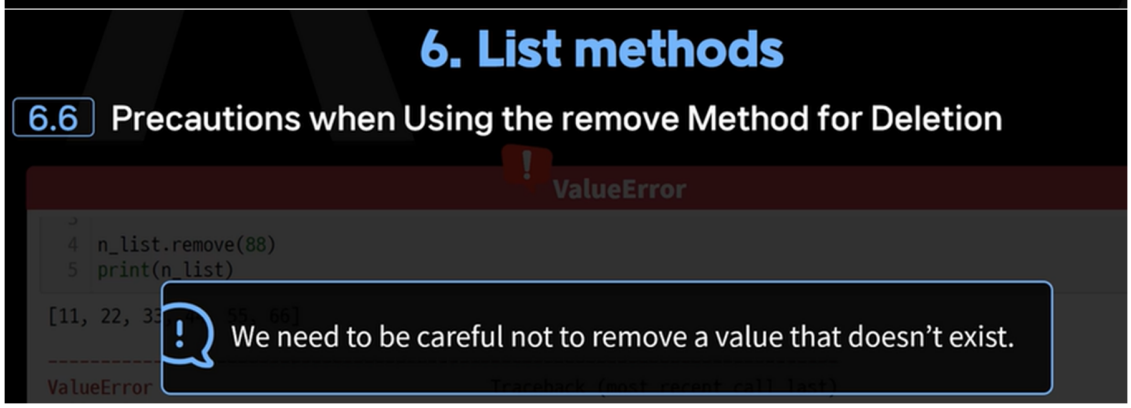
```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 n_list.remove(88)
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[42], line 4
      1 n_list = [11, 22, 33, 44, 55, 66]
      2 print(n_list)
----> 4 n_list.remove(88)
```


6. List methods
6.6 Precautions when Using the remove Method for Deletion
ValueError
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
4 n_list.re
5 print(n_li
[11, 22, 33, 44, 55, 66]
There are some things to be cautious about when using the remove method.


6. List methods
6.6 Precautions when Using the remove Method for Deletion
ValueError
4 n_list.remove(88)
5 print(n_list)
[11, 22, 33, 44, 55, 66]
ValueError Traceback (most recent call last)
Cell In[42], line 4
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
----> 4 n_list.remove(88)
5 print(n_list)
ValueError: list.remove(x): x not in list
If we try to remove a nonexistent item, an error will occur.


6. List methods
6.6 Precautions when Using the remove Method for Deletion
ValueError
4 n_list.remove(88)
5 print(n_list)
[11, 22, 33
ValueError
We need to be careful not to remove a value that doesn't exist.

6. List methods

6.6 Precautions when Using the remove Method for Deletion

- The in operator checks if an item is present in a list.
  It is safe to delete an item from the list only when this condition is true.

```
1 bts = ["V", "J-Hope", "Jungkook"]
2 if 'Suga' in bts:
3     bts.remove('Suga')
4 print(bts)
```

It is safe to delete an item from the 'bts' list only when the condition of 'in' operator is true.

```
['V', 'J-Hope', 'Jungkook']
```

6. List methods

6.7 The pop Method for Deleting List Items

- The pop method returns and removes the **last item** in the list.

```
1 n_list = [10, 20, 30]
2 print(n_list)
```

n_list = [10, 20, 30]

n_list before pop: | 10 | 20 | 30 |

```
[10, 20, 30]
```

```
1 n = n_list.pop()
2 print('n = ', n)
3 print('n_list =', n_list)
```

after n = n_list.pop

n_list after pop: | 10 | 20 |

```
n =  30
n_list = [10, 20]
```

6. List methods

6.8 Deleting List Items Using the del Keyword

- The del keyword can be used before list indexing to delete the item at the specified index.
- Note: **You cannot delete using del 44.**

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 del n_list[3]
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

6. List methods

6.8 Deleting List Items Using the del Keyword

- The del keyword can be used before list indexing to delete the item at the specified index.
- Note: **You cannot delete using del 44.**

```
1 n_list = [11, 22, 33, 44, 55, 66]
2 print(n_list)
3
4 del n_list[3]
5 print(n_list)
```

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

Line 4
- Using the index n_list[3], we can delete the fourth element, 44.

6. List methods

6.8 Deleting List Items Using the del Keyword

SyntaxError

```
1 bts = ["V", "J-hope", "Suga", "Jungkook"]
2 del bts[0]
3 bts
```

```
['J-hope', 'Suga', 'Jungkook']
```

del is a Python keyword that can be used to delete specific items from memory using an index.

```
1 bts[0].del
```

```
  Cell In[50], line 1
    bts[0].del
           ^
SyntaxError: invalid syntax
```

6. List methods

6.8 Deleting List Items Using the del Keyword

SyntaxError

```
1 bts = ["V", "J-hope", "Suga", "Jungkook"]
2 del bts[0]
3 bts
```

```
['J-hope', 'Suga', 'Jungkook']
```

```
1 bts[0].del
```

it is not a method of the list, so caution must be taken.

```
  Cell In[50], line 1
    bts[0].del
           ^
SyntaxError: invalid syntax
```

6. List methods

6.8 Deleting List Items Using the del Keyword

SyntaxError

```
1 bts = ["V", "J-hope", "Suga", "Jungkook"]
2 del bts[0]
3 bts
```

```
['J-hope', 'Suga', 'Jungkook']
```

```
1 bts[0].del
```

```
  Cell In[50], line 1
    bts[0].del
           ^
SyntaxError: invalid syntax
```

Using del as a method will result in a SyntaxError.

6. List methods

6.9 The sort Method for Sorting List Items

The sort method is used for sorting

- By default, it performs ascending order sorting.
- It is written as the dot (.) notation after the list variable.

Sorting techniques for lists

Ascending order sorting
Sorting in increasing order of values.

Descending order sorting
Sorting in increasing order of values.

6. List methods

6.9 The sort Method for Sorting List Items

- By using the sort method of a list, we can sort the list elements in ascending order.
- If the parameter reverse = True is given, it will sort the list in descending order.

```
1 list1 = [20, 10, 40, 50, 30]
2 list1.sort()
3 list1
```

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

```
1 list1.sort(reverse = True)
2 print(list1)
```

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

6. List methods

6.10 Various methods provided by lists

Method	What it does
index(x)	Finds the position using element x.
append(x)	Adds element x to the end of the list.
count(x)	Returns the number of occurrences of object x in the list.
extend([x1, x2])	Inserts items [x1, x2] into the list.
insert(index, x)	Adds x at the desired index position.

6. List methods

6.10 Various methods provided by lists

Method	What it does
remove(x)	Deletes the element x from the list.
pop(index)	Deletes and returns the element at the index. If index is omitted, it deletes and returns the last element of the list.
sort()	Sorts the values in ascending order. If the reverse parameter is set to True, it sorts in descending order.
reverse()	Reverses the list, changing the order of the original elements.

7. Built-in functions used with lists

7.1 Obtaining the Length, Maximum, and Sum of a List

- When we have a numerical list, we can conveniently utilize built-in functions such as len, max, min, sum, any, etc.

```
1 n_list = [200, 700, 500, 300, 400]
2 len(n_list)
```

```
5
```

len

- len returns the length of the list.

7. Built-in functions used with lists

7.1 Obtaining the Length, Maximum, and Sum of a List

- When we have a numerical list, we can conveniently utilize built-in functions such as len, max, min, sum, any, etc.

```
1 max(n_list)
```

```
700
```

```
1 min(n_list)
```

```
200
```

max / min
- max returns the maximum value among the list items, while min returns the minimum value.

7. Built-in functions used with lists

7.1 Obtaining the Length, Maximum, and Sum of a List

- When we have a numerical list, we can conveniently utilize built-in functions such as len, max, min, sum, any, etc.

```
1 sum(n_list)
```

```
2100
```

sum
- sum adds up all the elements in the list if they are numeric.

7. Built-in functions used with lists

7.2 Lists and the any Function

- Among the available functions for lists, the any function returns True if there is at least one non-zero element in the list.

```
1 n_list = [200, 700, 500, 300, 400]
2 a_list = [0, '']
```

```
1 any(n_list)
```

The function any(n_list) returns True.

```
True
```

```
1 any(a_list)
```

The list 'a_list', which consists of only 0 and '' (empty string), returns False when used with the any function.

```
False
```