1D array slicing:

1d array slicing refers to accessing a subset of elements from a one-dimensional array (or list) based on specified indices. In Python, slicing is performed using the colon : operator within square brackets [ ]. It allows us to extract parts of an array without modifying the original one.

Syntax for slicing: `array[start:stop:step]`

- start: Index to begin the slice (inclusive). Default is 0.
- stop: Index to end the slice (exclusive). Slicing stops at stop - 1. Default is the length of the array.
- step: Interval between elements. Default is 1.

Examples:

Suppose you have the following 1D array:

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

1. **Basic slicing:**
   ```
   arr[1:4]  # [20, 30, 40]
   ```

Starts at index 1 and ends before index 4.

2. **Omitting start:**
   ```
   arr[:3]  # [10, 20, 30]
   ```

Starts from the beginning of the array.

3. **Omitting stop:**
   ```
   arr[3:]  # [40, 50, 60]
   ```

Slices until the end of the array.

4. **Using a step:**
   ```
   arr[0:5:2]  # [10, 30, 50]
   ```

Includes every second element from index 0 to 4.

5. **Negative indices:**
   arr[-4:-1] # [30, 40, 50]

Indexing starts from the end (-1 is the last element).

6. **Reverse slicing:**
   arr[::-1] # [60, 50, 40, 30, 20, 10]

Reverses the array with a step of -1.

Key Points:

- Slicing does not modify the original array; it creates a new subarray.
- If step is negative, slicing proceeds in reverse order.
- Omitting all parameters (arr[:]) creates a shallow copy of the array.

Slicing is a powerful feature for array and list manipulation in Python, making it easy to access specific parts of data.

Coding:

 # 1d array slicing:

```
import numpy as np

arr = np.array([10,20,30,40,50,60,70])

# 1. Slice specific range of elements (from index 1 to 4, exclusive)

slice1 = arr[1:4]

# 2. Slice from the beginning to a specific index

slice2 = arr[:5]

# 3. Slice from a specific index to the end

slice3 = arr[2:]
```

```
# 4. Slice with a step of 2

slice4 = arr[0:5:2]

# 5. Slice using negative indices

slice5 = arr[-5:-1]

# 6. Reverse the array using slicing

slice6 = arr[::-1]

# Print all the slices

print("Original Array:", arr)

print("Slice 1 :", slice1)

print("Slice 2 :", slice2)

print("Slice 3 :", slice3)

print("Slice 4 :", slice4)

print("Slice 5 :", slice5)

print("Slice 6 :", slice6)
```

Output:

```
Original Array: [10 20 30 40 50 60 70]

Slice 1 : [20 30 40]

Slice 2 : [10 20 30 40 50]

Slice 3 : [30 40 50 60 70]

Slice 4 : [10 30 50]

Slice 5 : [30 40 50 60]
```

Slice 6 : [70 60 50 40 30 20 10]

2d array slicing:

2D array slicing refers to accessing specific subsets of rows and columns from a two-dimensional array or matrix. It allows efficient manipulation and extraction of data in a structured manner. The concept extends from 1D slicing, but with an additional dimension.

Syntax for slicing:

array[row_start:row_stop:row_step, col_start:col_stop:col_step]

- row_start:row_stop:row_step: Specifies the range and step for slicing rows.
- col_start:col_stop:col_step: Specifies the range and step for slicing columns.

Example:

Consider the following 2D array (matrix):

```
import numpy as np

arr = np.array([

    [1, 2, 3, 4],

    [5, 6, 7, 8],

    [9, 10, 11, 12],

    [13, 14, 15, 16]

])
```

1. Extracting specific rows and columns:

```
arr[1:3, 2:4]
```

- Selects rows 1 to 2 (index 3 is excluded). Selects columns 2 to 3 (index 4 is excluded). Output:

[[7, 8],

 [11, 12]]

2. Selecting specific rows:

arr[1:3, :]

- Rows 1 to 2 are selected.
- : selects all columns. Output:

[[5, 6, 7, 8],

 [9, 10, 11, 12]]

3. Selecting specific columns:

arr[:, 1:3]

- : selects all rows.
- Columns 1 to 2 are selected. Output:

[[2, 3],

 [6, 7],

 [10, 11],

 [14, 15]]

4. Using a step:

arr[::2, ::2]

- Rows are selected with a step of 2 (every other row).
- Columns are selected with a step of 2 (every other column). Output:

[[1, 3], [9, 11]]

5. Negative slicing for reverse order:

arr[::-1, ::-1]

Reverses rows and columns.

Output:

[[16, 15, 14, 13],

[12, 11, 10, 9],

[8, 7, 6, 5],

[4, 3, 2, 1]]

Key Points:

- **Omitting indices: Leaving out start, stop, or step defaults to the full range or step of 1.**
- **Combination: Slicing rows and columns independently provides flexibility for complex data extraction.**
- **Works with NumPy arrays: While you can slice lists of lists, NumPy arrays offer more efficiency and additional functionality.**

2D slicing is especially useful in numerical computing, data analysis, and image processing tasks where data is inherently multidimensional.

Coding:

 # 2d array slicing:

```
import numpy as np

arr = np.array([

    [1,2,3,4],

    [5,6,7,8],
```

```
    [9,10,11,12],
    [13,14,15,16]
])
# 1. Slice a submatrix (rows 1 to 2 and columns 2 to 3)
slice1 = arr[1:3, 2:4]
# 2. Slice all rows and specific columns (columns 1 to 3)
slice2 = arr[:, 1:3]
# 3. Slice specific rows and all columns (rows 1 to 2)
slice3 = arr[1:3, :]
# 4. Slice with a step (every 2nd row and every 2nd column)
slice4 = arr[::2, ::2]
# 5. Reverse the order of rows and columns
slice5 = arr[::-1, ::-1]
# 6. Extract a single row (row 2)
slice6 = arr[2, :]
# 7. Extract a single column (column 3)
slice7 = arr[:, 3]
# Print all slices
print("Original Array:", arr)
print("Slice 1:", slice1)
print("Slice 2:", slice2)
```

```
print("Slice 3:", slice3)

print("Slice 4:", slice4)

print("Slice 5:", slice5)

print("Slice 6:", slice6)

print("Slice 7:", slice7)
```

Output:

```
Original Array: [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Slice 1: [[ 7  8]
 [11 12]]
Slice 2: [[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
Slice 3: [[ 5  6  7  8]
 [ 9 10 11 12]]
Slice 4: [[ 1  3]
 [ 9 11]]
Slice 5: [[16 15 14 13]
```

[12 11 10 9]

[8 7 6 5]

[4 3 2 1]]

Slice 6: [9 10 11 12]

Slice 7: [4 8 12 16]

3d array slicing:

3D array slicing is a technique used to extract specific subsets of data from a three-dimensional array. A 3D array can be thought of as a stack of 2D arrays, and slicing allows you to select elements, rows, columns, or even entire layers based on your requirements.

Understanding a 3D Array:

A 3D array has three dimensions:

1. Depth (or layers): The "stack" of 2D arrays.
2. Rows: Horizontal slices in each layer.
3. Columns: Vertical slices in each layer.

Syntax of Slicing:

The slicing syntax for a 3D array is:

`array[start1:end1:step1, start2:end2:step2, start3:end3:step3]`

- First index: Refers to the depth (layers).
- Second index: Refers to the rows.
- Third index: Refers to the columns.

Slicing Components:

1. start: Starting index (inclusive).
2. end: Ending index (exclusive).
3. step: Step size for skipping elements.

Example with Explanation:

```
import numpy as np

# Create a 3D array (2 layers, 3 rows, 4 columns)
array = np.arange(24).reshape(2, 3, 4)
print("Original 3D Array:\n", array)

# Slice 1: Extract the first layer
slice1 = array[0, :, :]
print("\nFirst Layer:\n", slice1)

# Slice 2: Extract the second row from all layers
slice2 = array[:, 1, :]
print("\nSecond Row Across All Layers:\n", slice2)

# Slice 3: Extract elements from the first layer, rows 1 to 2, columns 2 to 3
slice3 = array[0, 1:3, 2:4]
print("\nSliced Portion:\n", slice3)
```

Explanation:

1. array[0, :, :]
    - 0: Selects the first layer.
    - :: Selects all rows.
    - :: Selects all columns.
    - Result: The entire first layer is returned.

2. array[:, 1, :]
    - :: Selects all layers.
    - 1: Selects the second row in each layer.
    - :: Selects all columns.
    - Result: The second row across all layers is returned.
3. array[0, 1:3, 2:4]
    - 0: Selects the first layer.
    - 1:3: Selects rows with indices 1 and 2.
    - 2:4: Selects columns with indices 2 and 3.
    - Result: A smaller subset from the first layer is returned.

Visualization:

For an array like this:

```
Layer 0:
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],
 [ 8,  9, 10, 11]]

Layer 1:
[[12, 13, 14, 15],
 [16, 17, 18, 19],
 [20, 21, 22, 23]]
```

- array[0, :, :] gives Layer 0.
- array[:, 1, :] gives [[4, 5, 6, 7], [16, 17, 18, 19]].
- array[0, 1:3, 2:4] gives [[6, 7], [10, 11]].

This demonstrates how you can extract meaningful portions of a 3D array.

Code:

```
# 3d array slicing:

import numpy as np
```

```
# Create a 3D array with shape (2, 3, 4) (2 layers, 3 rows, 4 columns)

array = np.arange(24).reshape(2, 3, 4)

print("original array is:", array)

# Example 1: Extract the first layer

slice1 = array[0, :, :]

# Example 2: Extract all layers but only the second row

slice2 = array[:, 1, :]

# Example 3: Extract all layers, all rows, and only the first two columns

slice3 = array[:, :, 0:2]

# Example 4: Extract specific portion from the first layer (rows 1 to 2, columns 2 to 3)

slice4 = array[0, 1:3, 2:4]

# Example 5: Extract every alternate row and column from all layers

slice5 = array[:, ::2, ::2]

print("slice1 is:", slice1)

print("slice2 is:", slice2)

print("slice3 is:", slice3)

print("slice4 is:", slice4)

print("slice5 is:", slice5)
```

Output:

```
original array is: [[[ 0  1  2  3]
```

[4 5 6 7]

[8 9 10 11]]

[[12 13 14 15]

[16 17 18 19]

[20 21 22 23]]]

slice1 is: [[0 1 2 3]

[4 5 6 7]

[8 9 10 11]]

slice2 is: [[4 5 6 7]

[16 17 18 19]]

slice3 is: [[[0 1]

[4 5]

[8 9]]

[[12 13]

[16 17]

[20 21]]]

slice4 is: [[6 7]

[10 11]]

slice5 is: [[[0 2]

[8 10]]

[[12 14]

[20 22]]]

4d array slicing:

4D array slicing is similar to slicing in lower-dimensional arrays but involves an additional dimension. A 4D array can be thought of as a collection of 3D arrays stacked together. Slicing in a 4D array allows you to select subsets of data across these four dimensions.

Understanding a 4D Array:

A 4D array can be represented with these dimensions:

1. Batch (or collections): Represents groups of 3D arrays.
2. Depth (or layers): Represents the stack of 2D arrays within each batch.
3. Rows: Represents the horizontal slices in each layer.
4. Columns: Represents the vertical slices in each layer.

Slicing Syntax for a 4D Array:

`array[start1:end1:step1, start2:end2:step2, start3:end3:step3, start4:end4:step4]`

- First index (start1:end1:step1): Refers to the batch or collection.
- Second index (start2:end2:step2): Refers to the depth or layers in each batch.
- Third index (start3:end3:step3): Refers to the rows.
- Fourth index (start4:end4:step4): Refers to the columns.

Example of a 4D Array in Python

```
import numpy as np


# Create a 4D array with shape (2 batches, 3 layers, 4 rows, 5 columns)

array = np.arange(120).reshape(2, 3, 4, 5)

print("Original 4D Array:\n", array)


# Example 1: Extract the first batch

slice1 = array[0, :, :, :]

print("\nFirst Batch:\n", slice1)


# Example 2: Extract the first layer from each batch

slice2 = array[:, 0, :, :]

print("\nFirst Layer from Each Batch:\n", slice2)


# Example 3: Extract all batches, layers 1 to 2, rows 2 to 3, columns 3 to 4

slice3 = array[:, 1:3, 2:4, 3:5]

print("\nSubset of Array (Layers 1 to 2, Rows 2 to 3, Columns 3 to 4):\n", slice3)


# Example 4: Extract every alternate layer, row, and column from all batches

slice4 = array[:, ::2, ::2, ::2]
```

```
print("\nEvery Alternate Layer, Row, and Column from All Batches:\n", slice4)
```

Output Explanation

`array = np.arange(120).reshape(2, 3, 4, 5)`

The array will have:

- 2 batches
- Each batch contains 3 layers
- Each layer has 4 rows and 5 columns

Results:

1. Example 1 (slice1): Extract the first batch:
    - Result: All layers, rows, and columns from the first batch.
2. Example 2 (slice2): Extract the first layer from each batch:
    - Result: All rows and columns from the first layer of each batch.
3. Example 3 (slice3): Extract layers 1 to 2, rows 2 to 3, and columns 3 to 4 from all batches:
    - Result: A smaller subset of the array focusing on a specific region.
4. Example 4 (slice4): Extract every alternate layer, row, and column from all batches:
    - Result: A downsampled version of the original array.

Visualization of Dimensions

Consider a smaller 4D array `array.shape = (2, 2, 3, 4)`:

- Batch 0:
    - Layer 0: A 3x4 matrix
    - Layer 1: A 3x4 matrix
- Batch 1:
    - Layer 0: A 3x4 matrix

- Layer 1: A 3x4 matrix

By specifying indices for each dimension, you can extract specific portions across batches, layers, rows, or columns, making 4D slicing extremely powerful for multidimensional data manipulation, such as in image processing or deep learning tasks.

Code:

```
# 4d array slicing:

import numpy as np

# Create a 4D array with shape (2 batches, 2 layers, 3 rows, 4 columns)

array = np.arange(48).reshape(2, 2, 3, 4)

print("original array is:", array)

# Example 1: Extract the first batch

slice1 = array[0, :, :, :]

# Example 2: Extract the first layer from each batch

slice2 = array[:, 0, :, :]

# Example 3: Extract rows 1 to 2 and columns 2 to 4 from all layers and batches

slice3 = array[:, :, 1:3, 2:5]

# Example 4: Extract every alternate layer, row, and column from all batches

slice4 = array[:, ::2, ::2, ::2]

# Example 5: Extract all batches, last layer, and last two rows and columns

slice5 = array[:, -1, -2:, -2:]

print("slice1 is:", slice1)
```

```
print("slice2 is:", slice2)

print("slice3 is:", slice3)

print("slice4 is:", slice4)

print("slice5 is:", slice5)
```

Output:

```
original array is: [[[[ 0  1  2  3]
   [ 4  5  6  7]
   [ 8  9 10 11]]


  [[12 13 14 15]
   [16 17 18 19]
   [20 21 22 23]]]



 [[[24 25 26 27]
   [28 29 30 31]
   [32 33 34 35]]


  [[36 37 38 39]
   [40 41 42 43]
   [44 45 46 47]]]]
```

```
slice1 is: [[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
slice2 is: [[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[24 25 26 27]
  [28 29 30 31]
  [32 33 34 35]]]
slice3 is: [[[[ 6  7]
   [10 11]]

  [[18 19]
   [22 23]]]
```

[[[30 31]

[34 35]]

[[42 43]

[46 47]]]]

slice4 is: [[[[0 2]

[8 10]]]

[[[24 26]

[32 34]]]]

slice5 is: [[[18 19]

[22 23]]

[[42 43]

[46 47]]]