Journal of Physics:
Conference Series

Exploring Indexing and Slicing in NumPy Arrays

Journal of Physics: Conference Series **2832** (2024) 012019 doi:10.1088/1742-6596/2832/1/012019

Exploring Indexing and Slicing in NumPy Arrays

Yangyang Li, Yue Shao, Chunlong Fu*

[1] Sichuan University Jinjiang College, Meishan, Sichuan Province, 620860, P.R.China

*Corresponding author's e-mail: fuchunlong@scujj.edu.cn

Abstract. This article provides an in-depth analysis of the application differences between indexing and slicing NumPy arrays to provide a clear guide to scientific computing and data analysis. The article first introduces the basics of NumPy arrays, discusses the deep copy and shallow copy mechanisms, and the role of copies and views in memory management. Then it shows the specific application methods of integer indexing, boolean indexing, fancy indexing, and slicing of one- and two-dimensional arrays through examples. It can be concluded from the experiments that slices are views of the original arrays, and modifications to the views affect the original arrays. While the index returns a copy of the original array, the modification of the copy does not affect the original array. Finally, the syntactic forms, differentiation methods and application scenarios of array indexes and slices are summarized. The research in this paper aims to be able to help researchers and developers better master the indexing and slicing techniques of NumPy arrays, and improve their ability to process and analyze data in practice.

1. Introduction

With the rapid development of data availability, arithmetic power, and new algorithms in recent years, machine learning has emerged as one of the core methods for realizing artificial intelligence (AI)[1] . In the field of machine learning, Python is the most commonly used programming language in the field of machine learning. Today there are several widely used libraries to support the application and development of this field. And the NumPy library, as an open source Python scientific computing library, plays a crucial role in the implementation of machine learning algorithms. It is now a tool that more and more developers are choosing to use[2,3,4] . In NumPy, the indexing and slicing functions in arrays play a very important role in the implementation of the code. However, since they are very similar in terms of presentation, they are easily confused, and will lead to errors in the final result. So mastering the use of indexing and slicing is an important part of being able to use NumPy proficiently. In this article, we will compare the differences between indexing and slicing based on an introduction to indexing and slicing techniques. Multi-dimensional arrays of indexing and slicing and other operations with one-dimensional and two-dimensional arrays are similar, this article will be one-dimensional and two-dimensional arrays as a representative of the study.

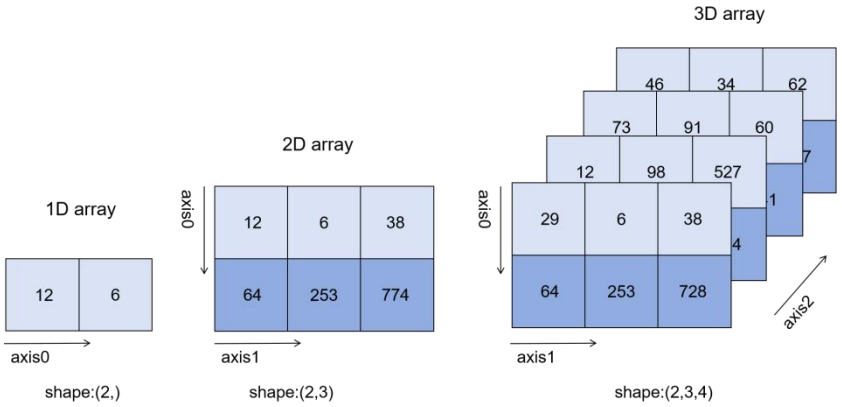

Figure 1. Multidimensional array

In numpy arrays, the concept of axis is closely related to dimension. When we refer to the dimension of an array, we are actually referring to each axis of the array. In Figure1, 1D array, 2D array and 3D array are shown respectively. For example, if the shape in 2D array is (2,3), then it has two axes. The first is 0 axis, which means the axis along the column direction, is vertically extended vertically across the rows, and the size is two. The second is 1 axis, which is the axis along the row direction, is horizontally extended horizontally across the columns, and the size is three. [5]

2. Shallow Copy and Deep Copy

In Python, copying is done to reuse some or all of the data of the original object, by creating a new object based on the original object using duplication. Deep copying and shallow copying are the two different ways used to copy an object [6,7] .

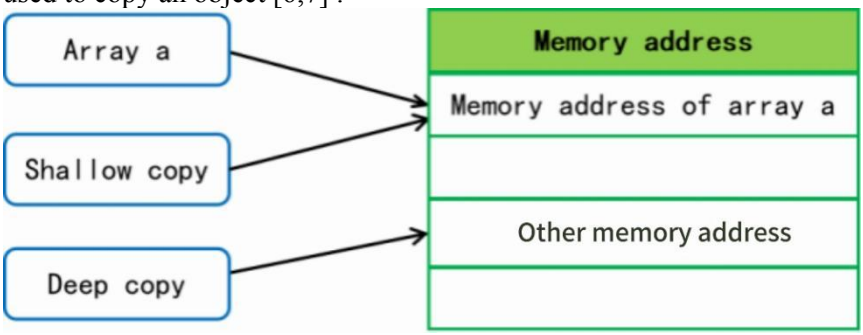

Figure 2. Shallow copy and Deep copy

As can be seen from Figure 2, the shallow copy memory address is the same as the original array, the data is stored in the same memory address. The deep copy memory address is different from the original array, the data is stored in the new memory address.

2.1. Shallow Copy

Shallow copy is a method of copying an object bitwise. It copies only the pointer to an object and creates a new object with all the attribute values of the original object. For basic data types, shallow copy directly gives a copy of the attribute values of the original data to the new member variable. For member variables of reference data types such as arrays, shallow copy passes the memory address of the member variable to the new member variable by reference. Both of them actually point to the same instance.

2.2. Deep Copy

Deep copy is another type of object copying that creates a new object and makes a full copy of the member variables of the referenced data type. For basic data types, deep copy is value passing, where a copy of the attribute value is given to the new member variable. For member variables of a referenced data type, such as an object of a particular class, deep copy requests new storage space. And will copy all the data of that referenced object and store it in a new memory space.

3. Copies and Views

In NumPy, copies and views are two different kinds of array objects. During array operations, the obtained array returned after array indexing is a copy. This copy can be regarded as a deep copy of the array and is separate from the original array in terms of memory address. The array returned after slicing is the view. The view can be thought of as a shallow copy of the array, sharing the data buffer with the original array.

3.1. Copies——np.copy( )

A copies is a complete copy of a piece of data, and if it is modified, it will not affect the original data. Variables obtained through copies do not interfere with each other, and a change in the value of one of them does not affect the values of the others.

3.2. Views——ndarray.view( )

A views is an alias or reference to the data reference, through the view can be realized for the original data access and operation. The view is not a copy of the original data, but a reference to the original data. It is obtained by slicing, mapping or other conversion operations on the original data.

4. Array Indexing

An array index is the way used to access and manipulate the elements of an array. In NumPy, square brackets are used to get the elements of an array (the index starts at 0). Array indexes include integer indexes, boolean indexes, and fancy indexes.[8]

4.1. Integer Indexing

Create X as a one-dimensional array [3,1,4,1,5,9,2,6] and Y as a two-dimensional array [[3,1,4,1],[5,9,2,6]]:

One-dimensional array: A one-dimensional array has only one dimension, so an integer can be used directly as an index.

Example: ① *`X[-2]` # Output: 2*

② *`X[3]` # Output: 1*

Two-dimensional array: A two dimensional array has two dimensions, and two integers can be used to represent rows and columns, respectively.

Example: ① *`Y[0, 2]` # Output: 4*

② *`Y[-1, 2]` # Output: 2*

4.2. Boolean Indexing

Boolean indexing is an efficient way of retrieving data that allows us to directly select data that meets specific conditions via a boolean array. The shape of the boolean array used for indexing must be the same as the initial size of the array to be indexed. And the result of boolean indexing is a one-dimensional array.

Example: *y1=Y>3*

Print(y1) #Output: array([[False,False,True,False],[True,True,False,True]])

Print(Y[y1]) #Output: array([4,5,9,6])

4.3. Fancy Indexing

Fancy indexing refers to indexing using integer arrays, where integerarrays can be NumPy arrays or iterable types like lists, tuples, etc. in Python.

For example, it is possible to use an array to index another array. In this case, the result will be a new array with the same shape as the indexed array. Each element of it is the value of the corresponding subscript in the indexed array.

Fancy indexing of one-dimensional arrays:

One-dimensional array fancy indexing:

Example: *x1=X[0,3,4]*

Print(x1) #Output:array（[3,1,5]）

x2=X[np.array([1,1],[2,3])]

Print(x2) #Output:array（[1,1],[4,1]）

Two-dimensional array fancy indexing:

Example: *print (Y[np.array([0,-1)]), np.array([1,2)])]) #Equivalent to Y[[0,-1],[1,2]]*

#Output: array([1,2])

4.4. Array Indexing and Copies

The array index returns a copy of the values in the original array, and modifications to the copy will not affect the original array, as exemplified by the previous X and Y arrays.

```
x3=X.copy() #deep copy
print(x3 is X) #Output: False
print(id(X)) #Output: 2428360779984
print(id(x3)) #Output: 2428360779362
print(X) #Output: [3,1,4,1,5,9,2,6]
print(x3) #Output: [3,1,4,1,5,9,2,6]
X[0]=6
print(X) #Output: [6,1,4,1,5,9,2,6]
print(x3) #Output: [3,1,4,1,5,9,2,6]
y2=Y[[0,-1],[1,2]] #index Output: array([1,2])
y2[1]=666
print(y2) #Output: array([1,666])
print(Y) #Output: array([[3,1,4,1],[5,9,2,6]])
```

As you can see from the above code, the data in the deep copy is completely independent. Although the values of the elements of array X and array x3 are identical, their id functions are not the same. They belong to two different memory addresses, so a change in either one does not affect the other.

5. Array Slicing

The basic structure of a one-dimensional array slice is Z[a:b:c]. The basic structure of a two-dimensional array slice is Z[a:b,c:d]. Where before the comma represents the subscript range of rows and after the comma represents the subscript range of columns. And the subscript ranges of both rows and columns are left-closed and right-open. If only the range of rows is specified, for example Z[:b,c:d], the first row is 0 by default. Similarly, if only the range of columns is specified, for example Z[a:b,:d], the first column is 0 by default.[9]

If a colon `:` is not used in a slice operation, it is considered an implicit slice. In some languages, this may mean fetching an entire sequence or array.

Example: implicit_slice = Y[0]

Print(implicit_slice) # Output: array([3,1,4,1])

Conversely if at least one colon `:` is used in the slicing operation, it is considered explicit slicing. This usually means that a start index, end index, and/or step size is specified in the slice.

Example: explicit_slice = Y[:,1:4]

Print(explicit_slice) # Output: array([[1,4,1],[9,2,6]])

In addition, you can use the built-in slice function and set the start, stop and step parameters for more complex slicing operations.

5.1. Slicing One-Dimensional Arrays

The slicing operation is performed by separating the slicing parameters X[start:stop:step] by colons (where X is a one-dimensional array [3,1,4,1,5,9,2,6]), Table 1 shows a case study of one-dimensional array slicing.

Table 1.One-Dimensional Array Slicing

Usage	Typical example	Result
[a:b:1]	X[1:8:2]	[1, 1, 9, 6]
[a:b]	X[1:3]	[1,4]
[a::2]	X[1::2]	[1, 1, 9, 6]
[:b:2]	X[:8:2]	[3, 4, 5, 2]
[a:]	X[1:]	[1,4,1,5,9,2,6]
[:b]	X[:-3]	[3,1,4,1,5]
[::c]	X[::2]	[3, 4, 5, 2]
[:]	X[:]	[3,1,4,1,5,9,2,6]

5.2. Slicing 2D arrays

Y[a,b] means to take a two-dimensional array, i.e., to take an N-dimensional array then there are N parameters, N-1 comma separated (where Y is a two-dimensional array [[3,1,4,1],[5,9,2,6]]), Table 2 is a case study of two-dimensional array slicing.

Table 2.Two-Dimensional Array Slicing

Usage	Typical example	Result
[a:b:c,d:e:f]	Y[0:2:1,0:4:2]	[[3,4],[5,2]]
[a:b,d:e]	Y[0:2,1:2]	[[1],[9]]
[a::c,d::f]	Y[0::1,1::2]	[[1,1],[9,6]]
[:b:c,:e:f]	Y[:2:1,:3:2]	[[3,4],[5,2]]
[a:,d:]	Y[1:,2:]	[[2, 6]]
[:b,:e]	Y[:2,:1]	[[3],[5]]
[::c,::f]	Y[::1,::2]	[[3,4],[5,2]]
[a:b,:]	Y[1:2,:]	[[5,9,2,6]]
[:,d:e]	Y[:,1:4]	[[1,4,1],[9,2,6]]
[0]	Y[0]	[3,1,4,1]
[:,:]	Y[:,:]	[[3,1,4,1],[5,9,2,6]]

5.3. Array Slicing and Views

Array slices are views of the original array, and modifications to the slices affect the original array. Where X is a one-dimensional array [3,1,4,1,5,9,2,6] and Y is a two-dimensional array [[3,1,4,1],[5,9,2,6]]:

```
x4=X #shallow copy
print(x4 is X) #Output: Ture
print(id(X)) #Output: 2428360779984
```

```
print(id(x4)) #Output: 2428360779984
X[2]=123
print(X) #Output: [3,1,123,1,5,9,2,6]
print(x4) #Output: [3,1,123,1,5,9,2,6]
x5=X.view() #X "shares view" with x5
x5.shape=(2,4)
X[1]=0
print(X) #Output: [3 0 4 1 5 9 2 6]
print(x5) #Output: [[3 0][4 1][5 9][2 6]]
y3=Y[0:2,1:2] #slice Output: array([[1],[9]])
y3[[0]]=888
print(y3) #Output: array([[888],[9]])
print(Y) #Output: array([[3,888,4,1],[5,9,2,6]])
```

The id function is used in python to get the memory address of an object. From the above code, it is clear that in a shallow copy, array X has the same memory address as array x4. Therefore, a change in array X will definitely cause a change in the value of the elements of array x4.

6. Comparison of Array Indexing and Slicing

In NumPy, array indexing and array slicing are formally very similar easy to confuse. However, there are differences in some syntactic forms, application scenarios and data handling. Here are the main differences between them:

6.1. Array Indexing

Definition: Array Indexing is the process of obtaining single or multiple elements at specific locations in an array. Usually the position of the element is not regular and there is no colon in the process.

Syntactic form: Use square brackets "[]" for array indexing. Integers, booleans, or other appropriate data types can be used for indexing.

Application Scenario: Array indexing is mainly used to get the elements at a specific position in an array. You can use a single index value to get a single element, or you can use multiple index values combined into a tuple to get multiple elements.

Data type: Indexes can be integer, boolean or fancy, depending on the user's needs and operations.

6.2. Array Slicing

Definition: Array slicing is the process of obtaining subarrays (elements in regular positions of an isochronous sequence) of an array; the obtaining process usually has a colon.Syntactic form: use colon ":" for array slicing. The slice includes a start index, an end index, and a step size.

Application Scenario: Array slicing is mainly used to get a portion of an array, either consecutive elements or elements selected in steps.

Data type: The result of slicing is still an array, sharing data with the original array, a view rather than a copy.

6.3. Summary of Differences

Return type: An index usually returns a single element of an array, while a slice usually returns an array containing multiple elements.

Data sharing: A slice is a view of the original array, modifications to the slice affect the original array, and the index returns a copy of the values in the original array.

Application Scenario: Indexes are used to get single or multiple specific elements, while slices are used to get subsets of arrays.

Overall, both indexing and slicing are important tools in NumPy, and the choice of which to use is based on specific tasks and needs.

7. Conclusion

This article provides an in-depth analysis of the application differences between indexing and slicing NumPy arrays to provide a clear guide to scientific computing and data analysis. The article first introduces the basics of NumPy arrays, and then explores the deep-copy vs. shallow-copy mechanism, as well as the role of copies and views in memory management. The main contribution of this article is to compare indexing and cuts in different situations. The use of integer indexing, boolean indexing, fancy indexing, and one- and two-dimensional array slicing is demonstrated with examples.

Through the analysis and study of array operations, it can be found that NumPy array indexing and slicing are similar in form though. However, they have obvious differences in syntactic forms, application scenarios and data processing. Indexing is used to obtain single or multiple elements at specific locations in an array. Slices are used to obtain a portion of an array, either consecutive elements or elements selected in steps. An index returns a copy of the values in the array; modifying the copy does not affect the original array. A slice is a view of the original array, and modifications to the slice affect the original array. Proper understanding and use of indexes and slices, then, is critical to efficient scientific computing and data analysis.

Through the theoretical analysis and verification of code cases arrived at the array operation belongs to the slicing or indexing of the judgment method, the summary rules are as follows:

1) Anything that contains a colon is a slice, where the colon can be explicit or implicit.
2) Anything that doesn't use a colon to fetch array data is an index, excluding the case of implicit colons.

The findings of this paper will be able to help researchers and developers to better master the indexing and slicing techniques of NumPy arrays. Improve their ability to process and analyze data in the real world. As the fields of data science and machine learning continue to evolve and the need for efficient data processing grows, mastering these basic yet critical techniques will be of great benefit.

References

[1] M. Yücel, Bekda G, Nigdeli S M .Review and Applications of Machine Learning and Artificial Intelligence in Engineering[J]. 2020
[2] Chen Z, Xiong P. RSOME in Python: An Open-Source Package for Robust Stochastic Optimization Made Easy[J]. INFORMS journal on computing, 2023(4):35.
[3] Walt S V D, Colbert S C, Varoquaux G .The NumPy Array: A Structure for Efficient Numerical Computation[J]. COMPUTING IN SCIENCE AND ENGINEERING, 2011
[4] Mckinney W. Data Structures for Statistical Computing in Python[J]. proc. Python sci. conf, 2010.
[5] Harris, Charles R., et al. "Array programming with NumPy." Nature 585.7825 (2020): 357-362.
[6] WEN X M, GAO L T, QI A H. Deep copy and shallow copy in object-oriented programming[J]. Fujian Computer, 2004(9):2.
[7] Luo F, Du R, Zeng Z, Li Y. Discussion on copy constructor in C++ programming language[J]. Proceedings of SPIE - The International Society for Optical Engineering, 2011, 8350:76.
[8] Appiah V. Python programming for biologists extract regions of Sequences using String Indexing and Slicing[J]. 2021.
[9] Chen Z, Chen L, Zhou Y. Dynamic Slicing of Python Programs[C]. 2014 IEEE 38th Annual Computer Software and Applications Conference. 2014.