NumPy Searching Arrays

Searching Arrays

You can search an array for a certain value, and return the indexes that get a match.

To search an array, use the where() method.

Example

Find the indexes where the value is 4:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
```

Example

Find the indexes where the values are even:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

Example

Find the indexes where the values are odd:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
x = np.where(arr%2 == 1)

print(x)
```

Search Sorted

There is a method called searchsorted() which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

The searchsorted() method is assumed to be used on sorted arrays.

Example

Find the indexes where the value 7 should be inserted:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

Search From the Right Side

By default the left most index is returned, but we can give side='right' to return the right most index instead.

Example

Find the indexes where the value 7 should be inserted, starting from the right:

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')

print(x)
```

Multiple Values

To search for more than one value, use an array with the specified values.

Example

Find the indexes where the values 2, 4, and 6 should be inserted:

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

NumPy Sorting Arrays

Sorting Arrays

Sorting means putting elements in an ordered sequence.

Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

The NumPy ndarray object has a function called sort(), that will sort a specified array.

Example

Sort the array:

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

Note: This method returns a copy of the array, leaving the original array unchanged.

You can also sort arrays of strings, or any other data type:

Example

Sort the array alphabetically:

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Example

Sort a boolean array:

```
import numpy as np

arr = np.array([True, False, True])

print(np.sort(arr))
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

Example

Sort a 2-D array:

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

NumPy Filter Array

Filtering Arrays
Getting some elements out of an existing array and creating a new array out of them is called filtering.

In NumPy, you filter an array using a boolean index list.

A boolean index list is a list of booleans corresponding to indexes in the array.

If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

Example
Create an array from the elements on index 0 and 2:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

Example

Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list

filter_arr = []

# go through each element in arr

for element in arr:
```

```
  # if the element is higher than 42, set the value to True, otherwise
False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

Example

Create a filter array that will return only even elements from the original array:

```
        import numpy as np
        arr = np.array([1, 2, 3, 4, 5, 6, 7])
        # Create an empty list
        filter_arr = []
        # go through each element in arr
        for element in arr:
  # if the element is completely divisble by 2, set the value to True,
otherwise False
          if element % 2 == 0:
            filter_arr.append(True)
          else:
```

```
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)

print(newarr)
```

Creating Filter Directly From Array

The above example is quite a common task in NumPy and NumPy provides a nice way to tackle it.

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to.

Example

Create a filter array that will return only values higher than 42:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)

print(newarr)
```

Example

Create a filter array that will return only even elements from the original array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

filter_arr = arr % 2 == 0
```

```
newarr = arr[filter_arr]

print(filter_arr)

print(newarr)
```