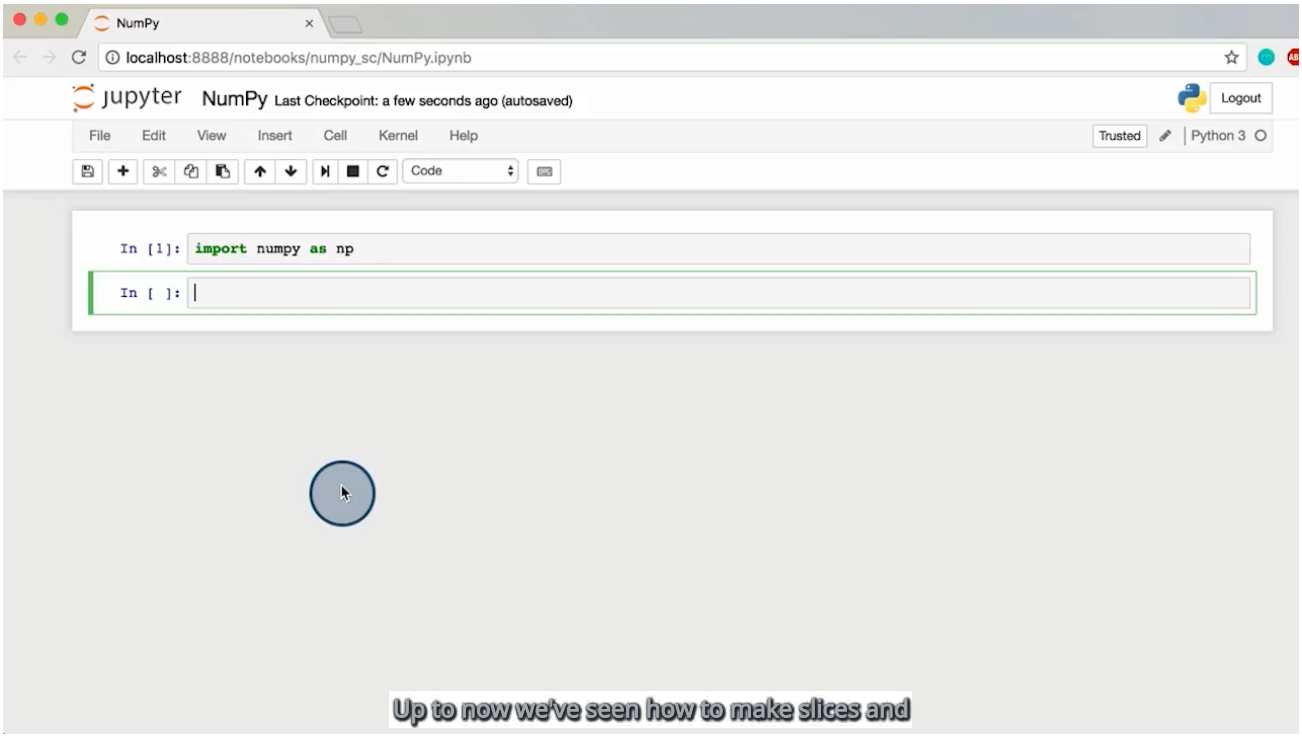
# 布尔型索引、集合运算和排序




00:00 / 03:14 1x CC

到目前为止，我们了解了如何使用索引进行切片以及选择 ndarray 元素。当我们知道要选择的元素的确切索引时，这些方法很有用。但是，在很多情况下，我们不知道要选择的元素的索引。例如，假设有一个 10,000 x 10,000 ndarray，其中包含从 1 到 15,000 的随机整数，我们只想选择小于 20 的整数。这时候就要用到*布尔型*索引，对于布尔型索引，我们将使用逻辑参数（而不是确切的索引）选择元素。我们来看一些示例：

```
# We print X
print()
print('Original X = \n', X)
print()

# We use Boolean indexing to select elements in X:
print('The elements in X that are greater than 10:', X[X > 10])
print('The elements in X that lees than or equal to 7:', X[X <= 7])
print('The elements in X that are between 10 and 17:', X[(X > 10) & (X < 17
)])

# We use Boolean indexing to assign the elements that are between 10 and 17 t
he value of -1
X[(X > 10) & (X < 17)] = -1

# We print X
print()
print('X = \n', X)
print()
```

```
Original X =
[[ 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]]
```

```
The elements in X that are greater than 10: [11 12 13 14 15 16 17 18 19 20 21 22
23 24]
The elements in X that lees than or equal to 7: [0 1 2 3 4 5 6 7]
The elements in X that are between 10 and 17: [11 12 13 14 15 16]
```

```
X =
[[ 0 1 2 3 4]
 [ 5 6 7 8 9]
 [10 -1 -1 -1 -1]
 [-1 -1 17 18 19]
 [20 21 22 23 24]]
```

```
# We create a rank 1 ndarray
x = np.array([1,2,3,4,5])

# We create a rank 1 ndarray
y = np.array([6,7,2,8,4])

# We print x
print()
print('x = ', x)

# We print y
print()
print('y = ', y)

# We use set operations to compare x and y:
print()
print('The elements that are both in x and y:', np.intersect1d(x,y))
print('The elements that are in x that are not in y:', np.setdiff1d(x,y))
print('All the elements of x and y:',np.union1d(x,y))
```

> x = [1 2 3 4 5]

> y = [6 7 2 8 4]

> The elements that are both in x and y: [2 4]
> The elements that are in x that are not in y: [1 3 5]
> All the elements of x and y: [1 2 3 4 5 6 7 8]

我们还可以在 NumPy 中对 ndarray 进行排序。我们将了解如何使用 `np.sort()` 函数以不同的方式对秩为 1 和 2 的 ndarray 进行排序。和我们之前看到的其他函数一样， `sort` 函数也可以当做方法使用。但是，对于此函数来说，数据在内存中的存储方式有很大变化。当 `np.sort()` 当做函数使用时，它不会对ndarray进行就地排序，即不更改被排序的原始 ndarray。但是，如果将 `sort` 当做方法， `ndarray.sort()` 会就地排序 ndarray，即原始数组会变成排序后的数组。我们来看一些示例：

```
# We print x
print()
print('Original x = ', x)

# We sort x and print the sorted array using sort as a function.
print()
print('Sorted x (out of place):', np.sort(x))

# When we sort out of place the original array remains intact. To see this we
 print x again
print()
print('x after sorting:', x)
```

> Original x = [9 6 4 4 9 4 8 4 4 7]

> Sorted x (out of place): [4 4 4 4 4 6 7 8 9 9]

> x after sorting: [9 6 4 4 9 4 8 4 4 7]

注意， `np.sort()` 会对数组进行排序，但是如果被排序的 ndarray 具有重复的值， `np.sort()` 将在排好序的数组中保留这些值。但是，我们可以根据需要，同时使用 sort 函数和 unique 函数仅对 `x` 中的*唯*一元素进行排序。我们来看看如何对上述 `x` 中的唯一元素进行排序：

```
# We sort x but only keep the unique elements in x
print(np.sort(np.unique(x)))
```

> [4 6 7 8 9]

最后，我们来看看如何将 sort 当做方法，原地对 ndarray 进行排序：

```
# We print x
print()
print('Original x = ', x)

# We sort x and print the sorted array using sort as a method.
x.sort()

# When we sort in place the original array is changed to the sorted array. To
 see this we print x again
print()
print('x after sorting:', x)
```

> Original x = [9 9 8 1 1 4 3 7 2 8]

> x after sorting: [1 1 2 3 4 7 8 8 9 9]

在对秩为 2 的 ndarray 进行排序时，我们需要在 `np.sort()` 函数中指定是按行排序，还是按列排序。为此，我们可以使用关键字 `axis` 。我们来看一些示例：

```
# We create an unsorted rank 2 ndarray
X = np.random.randint(1,11,size=(5,5))

# We print X
print()
print('Original X = \n', X)
print()

# We sort the columns of X and print the sorted array
print()
print('X with sorted columns :\n', np.sort(X, axis = 0))

# We sort the rows of X and print the sorted array
print()
print('X with sorted rows :\n', np.sort(X, axis = 1))
```

> Original X =
> [[6 1 7 6 3]
>  [3 9 8 3 5]
>  [6 5 8 9 3]
>  [2 1 5 7 7]
>  [9 8 1 9 8]]

```
 [3 1 5 6 3]
 [6 5 7 7 5]
 [6 8 8 9 7]
 [9 9 8 9 8]]
```

```
X with sorted rows :
[[1 3 6 6 7]
 [3 3 5 8 9]
 [3 5 6 8 9]
 [1 2 5 7 7]
 [1 8 8 9 9]]
```

下一项