numpy index slice with None

Working through a sliding-window example for numpy. Was trying to understand the `,None` of `start_idx = np.arange(B[0])[:,None]`

```
foo = np.arange(10)
print foo
print foo[:]
print foo[:,]
print foo[:,None]
```

The effect of the `None` seems to be to transpose the array.

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
```

But I'm not totally sure. I haven't been able to find the documentation that explains what the second parameter (`None`) does. It's a hard fragment to google for, too. The numpy array docs makes me think it has something to do with advanced indexing, but I'm not sure enough.

python numpy

edited May 23 at 12:09
Community ♦
1 1

asked Nov 13 '16 at 14:35
user3556757
778 4 20

1 It's used as an alias for `np.newaxis` here being used to add singleton dimensions (dims with length=1). In the sample case, it was added to convert the input 1D array to `2D` with the second dim being singleton. – Divakar Nov 13 '16 at 14:38

1 Answer

`foo[:, None]` extends the 1 dimensional array `foo` into the second dimension. In fact, `numpy` uses the alias `np.newaxis` to do this.

consider `foo`

```
foo = np.array([1, 2])
print(foo)
```

```
[1 2]
```

A one dimensional array has limitations. For example, what's the transpose?

```
print(foo.T)
```

```
[1 2]
```

The same as the array itself

```
[ True True]
```

This limitation has many implications and it becomes useful to consider `foo` in higher dimensional context. numpy uses `np.newaxis`

```
print(foo[np.newaxis, :])
```

```
[[1 2]]
```

But this `np.newaxis` is just syntactic sugar for `None`

```
np.newaxis is None
```

```
True
```

So, often we use `None` instead because it's less characters and means the same thing

```
print(foo[None, :])
```

```
[[1 2]]
```

Ok, let's see what else we could've done. Notice I used the example with `None` in the first position while OP use the second position. This position specifies which dimension is extended. And we could've taken that further. Let these examples help explain

```
print(foo[None, :])  # same as foo.reshape(1, 2)
```

```
[[1 2]]
```

```
print(foo[:, None])  # same as foo.reshape(2, 1)
```

```
[[1]
 [2]]
```

```
print(foo[None, None, :])  # same as foo.reshape(1, 1, 2)
```

```
[[[1 2]]]
```

```
print(foo[None, :, None])  # same as foo.reshape(1, 2, 1)
```

```
[[[1]
  [2]]]
```

```
print(foo[:, None, None])  # same as foo.reshape(2, 1, 1)
```

```
[[[1]]

 [[2]]]
```

Keep in mind which dimension is which when numpy prints the array

```
print(np.arange(27).reshape(3, 3, 3))
```

```
          dim2
          ————————→
dim0 →  [[[ 0  1  2]   | dim1
          [ 3  4  5]   |
          [ 6  7  8]]  ↓
          ————————→
     →   [[ 9 10 11]   |
          [12 13 14]   |
          [15 16 17]]  ↓
          ————————→
     →   [[18 19 20]   |
          [21 22 23]   |
          [24 25 26]]] ↓
```

edited Nov 13 '16 at 15:15

answered Nov 13 '16 at 14:59

piRSquared

94.1k 10 51 131