Indexing

Index into NumPy arrays to extract data and array slices.

Chapter Goals:

- Learn about indexing arrays in NumPy
- Write code for indexing and slicing arrays

A. Array accessing

Accessing NumPy arrays is identical to accessing Python lists. For multi-dimensional arrays, it is equivalent to accessing Python lists of lists.

The code below shows example accesses of NumPy arrays.

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

arr = np.array([[6, 3], [0, 2]])
# Subarray
print(repr(arr[0]))
```

B. Slicing

NumPy arrays also support slicing. Similar to Python, we use the colon operator (i.e. `arr[:]`) for slicing. We can also use negative indexing to slice in the backwards direction.

The code below shows example slices of a 1-D NumPy array.

```
arr = np.array([1, 2, 3, 4, 5])
print(repr(arr[:]))
print(repr(arr[1:]))
print(repr(arr[2:4]))
print(repr(arr[:-1]))
print(repr(arr[-2:]))
```

For multi-dimensional arrays, we can use a comma to separate slices across each dimension.

The code below shows example slices of a 2-D NumPy array.

```
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
print(repr(arr[:]))
print(repr(arr[1:]))
print(repr(arr[:, -1]))
print(repr(arr[:, 1:]))
print(repr(arr[0:1, 1:]))
print(repr(arr[0, 1:]))
```

C. Argmin and argmax

In addition to accessing and slicing arrays, it is useful to figure out the actual indexes of the minimum and maximum elements. To do this, we use the `np.argmin` and `np.argmax` functions.

The code below shows example usages of `np.argmin` and `np.argmax`. Note that the index of element `-6` is index `5` in the flattened version of `arr`.

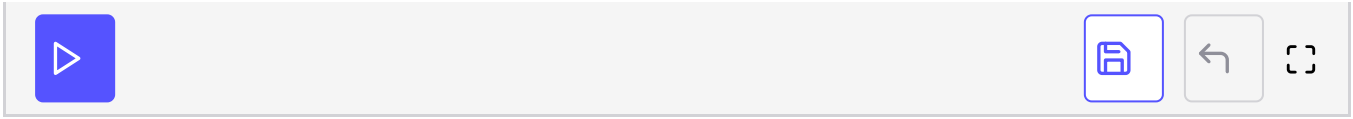
```
arr = np.array([[-2, -1, -3],
                [4, 5, -6],
                [-3, 9, 1]])
print(np.argmin(arr[0]))
print(np.argmax(arr[2]))
print(np.argmin(arr))
```

The `np.argmin` and `np.argmax` functions take the same arguments. The required argument is the input array and the `axis` keyword argument specifies which dimension to apply the operation on.

The code below shows how the `axis` keyword argument is used for these

functions.

```
arr = np.array([[-2, -1, -3],
                [4, 5, -6],
                [-3, 9, 1]])
print(repr(np.argmin(arr, axis=0)))
print(repr(np.argmin(arr, axis=1)))
print(repr(np.argmax(arr, axis=-1)))
```



In our example, using `axis=0` meant the function found the index of the minimum *row* element for each column. When we used `axis=1`, the function found the index of the minimum *column* element for each row.

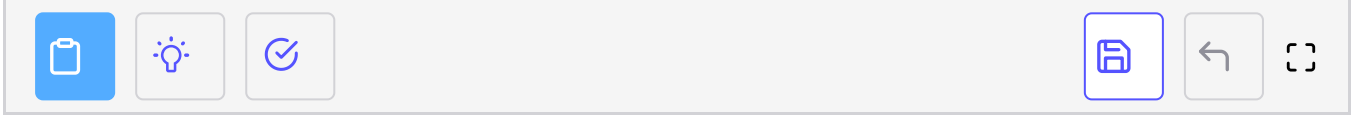
Setting `axis` to -1 just means we apply the function across the last dimension. In this case, `axis=-1` is equivalent to `axis=1`.

Time to Code!

Each coding exercise in this chapter will be to complete a small function that takes in a 2-D NumPy matrix (`data`) as input. The first function to complete is `direct_index`.

Set `elem` equal to the third element of the second row in `data` (remember that the first row is index 0). Then return `elem`.

```
def direct_index(data):
  # CODE HERE
  pass
```



The next function, `slice_data`, will return two slices from the input `data`.

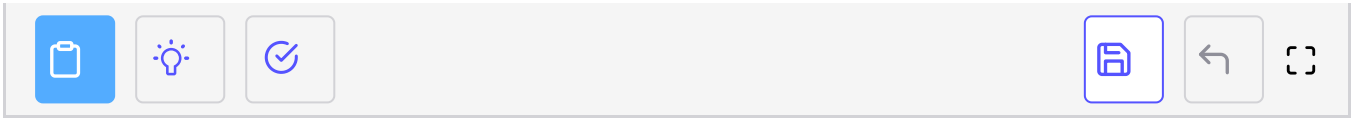
The first slice will contain all the rows, but will skip the first element in each row. The second slice will contain all the elements of the first three rows except the last two elements.

Set `slice1` equal to the specified first slice. Remember that NumPy uses a comma to separate slices along different dimensions.

Set `slice2` equal to the specified second slice.

Return a tuple containing `slice1` and `slice2`, in that order.

```
def slice_data(data):
  # CODE HERE
  pass
```



The next function, `argmin_data`, will find minimum indexes in the input `data`.

We can use `np.argmin` to find minimum points in the `data` array. First, we'll find the index of the overall minimum element.

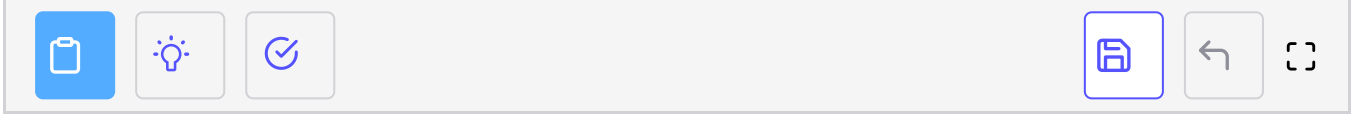
We can also return the indexes of each row's minimum element. This is equivalent to finding the minimum column for each row, which means our operation is done along axis `1`.

Set `argmin_all` equal to `np.argmin` with `data` as the only argument.

Set `argmin1` equal to `np.argmin` with `data` as the first argument and the specified `axis` keyword argument.

Return a tuple containing `argmin_all` and `argmin1`, in that order.

```
def argmin_data(data):
  # CODE HERE
  pass
```



The final function, `argmax_data`, will find the index of each row's maximum element in `data`. Since there are only 2 dimensions in `data`, we can apply the operation along either axis `1` or `-1`.

Set `argmax_neg1` equal to `np.argmax` with `data` as the first argument and `-1` as the `axis` keyword argument. Then return `argmax_neg1`.

```
def argmax_data(data):
  # CODE HERE
```

```
pass
```