Multidimensional arrays All Versions

Examples

Lists in lists

A good way to visualize a 2d array is as a list of lists. Something like this:

```
lst=[[1,2,3],[4,5,6],[7,8,9]]
```

here the outer list `lst` has three things in it. each of those things is another list: The first one is: `[1,2,3]` , the second one is: `[4,5,6]` and the third one is: `[7,8,9]` . You can access these lists the same way you would access another other element of a list, like this:

```
print (lst[0])
#output: [1, 2, 3]

print (lst[1])
#output: [4, 5, 6]

print (lst[2])
#output: [7, 8, 9]
```

You can then access the different elements in each of those lists the same way:

```
print (lst[0][0])
#output: 1

print (lst[0][1])
#output: 2
```

Here the first number inside the `[]` brackets means get the list in that position. In the above example we used the number `0` to mean get the list in the 0th position which is `[1,2,3]` . The second set of `[]` brackets means get the item in that position from the inner list. In this case we used both `0` and `1` the 0th position in the list we got is the number `1` and in the 1st position it is `2`

You can also set values inside these lists the same way:

```
lst[0]=[10,11,12]
```

Now the list is `[[10,11,12],[4,5,6],[7,8,9]]` . In this example we changed the whole first list to be a completely new list.

```
lst[1][2]=15
```

Now the list is `[[10,11,12],[4,5,15],[7,8,9]]` . In this example we changed a single element inside of one of the inner lists. First we went into the list at position 1 and changed the element within it at position 2, which was 6 now it's 15.

Lists in lists in lists in...

This behaviour can be extended. Here is a 3-dimensional array:

```
[[[111,112,113],[121,122,123],[131,132,133]],[[211,212,213],[221,222,223],[231,232,233]],[[311,3
```

As is probably obvious, this gets a bit hard to read. Use backslashes to break up the different dimensions:

```
[[[111,112,113],[121,122,123],[131,132,133]],\
 [[211,212,213],[221,222,223],[231,232,233]],\
 [[311,312,313],[321,322,323],[331,332,333]]]
```

By nesting the lists like this, you can extend to arbitrarily high dimensions.

Accessing is similar to 2D arrays:

```
print(myarray)
print(myarray[1])
print(myarray[2][1])
print(myarray[1][0][2])
etc.
```

And editing is also similar:

```
myarray[1]=new_n-1_d_list
myarray[2][1]=new_n-2_d_list
myarray[1][0][2]=new_n-3_d_list #or a single number if you're dealing with 3D arrays
etc.
```

Syntax

Parameters

Remarks