Lecture 12, 15 September 2022

Using numpy

- Arrays and lists
- Arrays are "homogenous" with regular structure
- Lists are flexible

Load numpy

In [31]:
```
import numpy as np
```

Constructing arrays

`np.array()` constructs an array from an input sequence

- Sequence can be a list, tuple, output of a `range()` command ...
- Size of the array is fixed by the sequence
- Underlying type is also fixed

In [32]:
```
a = np.array([1,2,3])
a
```

Out[32]:
```
array([1, 2, 3])
```

In [33]:
```
a = np.array((1,2,3))
a
```

Out[33]:
```
array([1, 2, 3])
```

In [34]:
```
b = np.array(range(10))
b
```

Out[34]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Use nested sequences to produce multi-dimensional arrays

- A 2d array is an array of 1d arrays
- Note: can mix and match notation for sequences, but dimensions much match

In [35]:
```
c = np.array([(0,1,0),[2,3,2]])
c
```

Out[35]:
```
array([[0, 1, 0],
       [2, 3, 2]])
```

In [36]:
```
cproblem = np.array([(0,1),[2,3,2]])
```

```
/home/madhavan/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: VisibleDeprecationWarning: Creating an ndarray
from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is de
precated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  """Entry point for launching an IPython kernel.
```

- A 3d array is an array of 2d arrays

In [37]:
```
d = np.array([[(0,1,0),[2,3,2]],[[4,5,4],[6,7,6]]])
d
```

Out[37]:
```
array([[[0, 1, 0],
        [2, 3, 2]],

       [[4, 5, 4],
        [6, 7, 6]]])
```

Pointwise scalar operations

In [38]:
```
a = np.arange(10)  # arange(n) is same as array(range(n))
a
```

Out[38]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [39]:
```
a**3  # Replace each element by its cube
```

Out[39]:
```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

In [40]:
```
a+3   # Add 3 to each element
```

Out[40]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [41]:
```
3*a   # Multiply each element by 3
```

Out[41]:
```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

Indexing and slicing

In [42]:
```
a[2], a[2:5]
```

Out[42]:
```
(2, array([2, 3, 4]))
```

Slice update

In [43]:
```
a[:6:2] = -1000  # equivalent to a[0:6:2] = -1000
a
```

Out[43]:
```
array([-1000,     1, -1000,     3, -1000,     5,     6,     7,     8,
           9])
```

Populate an array from a function

- Index is implicitly used as argument to the function

In [44]:
```
def f(x,y):
    return(10*x +  y)
```

In [45]:
```
f(5,7)
```

Out[45]:
```
57
```

In [46]:
```
b = np.fromfunction(f,(5,4),dtype=int)
b
```

Out[46]:
```
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
```

Indexing multi-dimensional arrays

In [47]:
```
b[2,3]  # Not b[2][3]
```

Out[47]:
```
23
```

In [48]:
```
b[0:5, 1] # second column in each row of b
```

Out[48]:
```
array([ 1, 11, 21, 31, 41])
```

In [49]:
```
b[ : ,1]  # equivalent to the previous example
```

Out[49]:
```
array([ 1, 11, 21, 31, 41])
```

In [50]:
```
b[1:3, :]  # all columns in the second and third row of b
```

Out[50]:
```
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

In [51]:
```
b[1:4,1:3] # extract a rectangular submatrix
```

Out[51]:
```
array([[11, 12],
       [21, 22],
       [31, 32]])
```

Iterating over elements

In [52]:
```
print(b)
```

```
[[ 0  1  2  3]
 [10 11 12 13]
 [20 21 22 23]
 [30 31 32 33]
 [40 41 42 43]]
```

In [53]:
```
for row in b:
    print(row)
```

```
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

In [54]:
```
for element in b.flat:
    print(element,end=' ')
```

```
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

Arrays and types

- When we pass a sequence of values, type is taken from the values we pass
- Default is float64, 64-bit float
- `np.zeros()` creates a zero array of required dimensions, but what "type" of () do we get?

In [55]:
```
a = np.zeros((5,7))
a
```

Out[55]:
```
array([[0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.]])
```

If we want `int` zeroes, we pass the desired type as an argument

In [56]:
```
a = np.zeros((5,7),dtype=int)
a
```

Out[56]:
```
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

Stacking arrays

In [57]:
```
a = np.floor(10*np.random.random((2,2)))
b = np.floor(10*np.random.random((2,2)))
print(a)
print(b)
```

```
[[5. 6.]
 [2. 5.]]
[[5. 5.]
 [6. 7.]]
```

`vstack` stacks a sequence of arrays vertically -- should have same number of columns

In [58]:
```
np.vstack((a,b))
```

Out[58]:
```
array([[5., 6.],
       [2., 5.],
       [5., 5.],
       [6., 7.]])
```

In [59]:
```
c = np.floor(10*np.random.random((3,3)))
c
```

Out[59]:
```
array([[1., 3., 7.],
       [2., 2., 8.],
       [4., 1., 7.]])
```

In [60]:
```
np.vstack((a,c))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_328462/3667487255.py in <module>
----> 1 np.vstack((a,c))

<__array_function__ internals> in vstack(*args, **kwargs)

~/miniconda3/lib/python3.7/site-packages/numpy/core/shape_base.py in vstack(tup)
    280     if not isinstance(arrs, list):
    281         arrs = [arrs]
--> 282     return _nx.concatenate(arrs, 0)
    283
    284

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 1, the array at index 0 has size 2 and the array at index 1 has size 3
```

In [61]:
```
d = np.floor(10*np.random.random((3,2)))
d
```

Out[61]:
```
array([[7., 0.],
       [6., 8.],
       [4., 1.]])
```

In [62]:
```
np.vstack((a,d))
```

Out[62]:
```
array([[5., 6.],
       [2., 5.],
       [7., 0.],
       [6., 8.],
       [4., 1.]])
```

Can stack any length sequence of arrays, not just two arrays

In [63]:
```
np.vstack([a,b,d])
```

Out[63]:
```
array([[5., 6.],
       [2., 5.],
       [5., 5.],
       [6., 7.],
       [7., 0.],
       [6., 8.],
       [4., 1.]])
```

Likewise, `hstack` stacks horizontally, number of rows must match

In [64]:
```
np.hstack((a,b))
```

Out[64]:
```
array([[5., 6., 5., 5.],
       [2., 5., 6., 7.]])
```

In [65]:
```
np.hstack((b,c))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_328462/2608850110.py in <module>
----> 1 np.hstack((b,c))

<__array_function__ internals> in hstack(*args, **kwargs)

~/miniconda3/lib/python3.7/site-packages/numpy/core/shape_base.py in hstack(tup)
    343         return _nx.concatenate(arrs, 0)
    344     else:
--> 345         return _nx.concatenate(arrs, 1)
    346
    347

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 0, the array at
index 0 has size 2 and the array at index 1 has size 3
```

In [66]:
```
e = np.floor(10*np.random.random((2,3)))
e
```

Out[66]:
```
array([[3., 2., 4.],
       [3., 1., 3.]])
```

In [67]:
```
np.hstack((a,b,e))
```

Out[67]:
```
array([[5., 6., 5., 5., 3., 2., 4.],
       [2., 5., 6., 7., 3., 1., 3.]])
```