Numpy Basics by Mrittika Megaraj

Numpy

```
List : Properties
Heterogenous
Mutable
Numpy : Data Structure
properties
1 ndimensional arrays
2.Homogenous
3.Mutable
4. Vectorization
5. Broadcasting is possible

Creation
1. Type Casting using np.array() method
2 . arange(start:Stop:Step),linspace(start:Stop, num=50),
random.random(): To generate random numbers between 0 and 1
random.randint() : To generate random integers between the starting
value and stop value
np.full()
reshape()
Transpose()
np.zeros
np.ones()

Help : Shift+Tab
Tab
```

In [1]:
```
l1=[1,2,3,4,5]
```

In [2]:
```
l1+5
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
Cell In[2], line 1
----> 1 l1+5

TypeError: can only concatenate list (not "int") to list
```

In [3]:
```
l1/2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
t)
Cell In[3], line 1
----> 1 l1/2

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

In [4]:
```
import numpy as np
```

In [5]:
```
ar1=np.array([1,2,3,4,5])
```

In [6]:
```
type(ar1)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
ar1
```

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

In [8]:
```
# 2D array
ar_2D=np.array([[1,2,3,4,5],[6,7,8,9,10]])
```

In [9]:
```
ar_2D
```

Out[9]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

In [10]:
```
l1
```

Out[10]:
```
[1, 2, 3, 4, 5]
```

In [11]:
```
ar2=np.array(l1)
```

In [12]:
```
type(ar2)
```

Out[12]:
```
numpy.ndarray
```

In [13]:
```
t1=(1,2,3,4,5)
```

In [14]:
```
type(t1)
```

Out[14]:
```
tuple
```

In [15]:
```
ar3=np.array(t1)
```

In [16]:
```
type(ar3)
```

Out[16]:
```
numpy.ndarray
```

In [17]:
```
ar3
```

Out[17]:
```
array([1, 2, 3, 4, 5])
```

In [18]:
```
print(dir(ar1))
```

```
['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__',
'__array_function__', '__array_interface__', '__array_prepare__', '__arra
y_priority__', '__array_struct__', '__array_ufunc__', '__array_wrap__',
'__bool__', '__class__', '__class_getitem__', '__complex__', '__contains_
_', '__copy__', '__deepcopy__', '__delattr__', '__delitem__', '__dir__',
'__divmod__', '__dlpack__', '__dlpack_device__', '__doc__', '__eq__', '__
float__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__
getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__iand__',
'__ifloordiv__', '__ilshift__', '__imatmul__', '__imod__', '__imul__', '_
_index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__i
or__', '__ipow__', '__irshift__', '__isub__', '__iter__', '__itruediv__',
'__ixor__', '__le__', '__len__', '__lshift__', '__lt__', '__matmul__', '_
_mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce
_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmatmul__', '__rmo
d__', '__rmul__', '__ror__', '__rpow__', '__rrshift__', '__rshift__', '__
rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem__', '__set
state__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__true
div__', '__xor__', 'all', 'any', 'argmax', 'argmin', 'argpartition', 'arg
sort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'con
j', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum', 'data', 'diagona
l', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten',
'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max', 'mean', 'min',
'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp',
'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchso
rted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', 'std',
'strides', 'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tos
tring', 'trace', 'transpose', 'var', 'view']
```

In [19]:
```
ar4=np.arange(0,21,2)
ar4
```

Out[19]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [20]:
```
ar5=np.linspace(1,20,10,dtype=int)
ar5
```

Out[20]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 20])
```

In [21]:
```
np.random.random(20)
```

Out[21]:
```
array([0.41750976, 0.88753088, 0.75400892, 0.67281514, 0.83596342,
       0.3219251 , 0.52425191, 0.08843268, 0.07214202, 0.35478599,
       0.01432808, 0.12397409, 0.14478798, 0.81881406, 0.21691611,
       0.68076671, 0.09984738, 0.59979366, 0.34438947, 0.85416865])
```

In [22]:
```
np.random.randint(1,20,10)
```

Out[22]:
```
array([ 4,  3,  2,  5,  9,  3, 13,  8, 19, 15])
```

In [23]:
```
ar1
```

Out[23]:
```
array([1, 2, 3, 4, 5])
```

In [24]:
```
ar5=np.arange(1,21)
```

In [25]:
```
ar5
```

Out[25]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 1
7,
       18, 19, 20])
```

In [26]:
```
ar5_2D=np.reshape(ar5,(5,4))
```

In [27]:
```
ar5_2D
```

Out[27]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [28]:
```
ar5_3D=np.reshape(ar5,(2,5,2))
```

In [29]:
```
ar5_3D
```

Out[29]:
```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6],
        [ 7,  8],
        [ 9, 10]],

       [[11, 12],
        [13, 14],
        [15, 16],
        [17, 18],
        [19, 20]]])
```

In [30]:
```
ar5_2D
```

Out[30]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [31]:
```
np.transpose(ar5_2D)
```

Out[31]:
```
array([[ 1,  5,  9, 13, 17],
       [ 2,  6, 10, 14, 18],
       [ 3,  7, 11, 15, 19],
       [ 4,  8, 12, 16, 20]])
```

In [32]:
```
np.arange()
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
Cell In[32], line 1
----> 1 np.arange()

TypeError: arange() requires stop to be specified.
```

In [33]:
```
ar5.
```

```
  Cell In[33], line 1
    ar5.
        ^
SyntaxError: invalid syntax
```

Inspection of arrays

```
.shape
.size
.ndim
.dtypes
.nbytes
```

In [34]:
```
ar5_2D.shape
```

Out[34]:
```
(5, 4)
```

In [35]:
```
ar5_2D.size
```

Out[35]:
```
20
```

In [36]:
```
ar5_2D
```

Out[36]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [37]:
```
ar5_2D.ndim
```

Out[37]:
```
2
```

In [38]:
```
ar1
```

Out[38]:
```
array([1, 2, 3, 4, 5])
```

In [39]:
```
ar1.ndim
```

Out[39]:
```
1
```

In [40]:
```
ar5_3D
```

Out[40]:
```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6],
        [ 7,  8],
        [ 9, 10]],

       [[11, 12],
        [13, 14],
        [15, 16],
        [17, 18],
        [19, 20]]])
```

In [41]:
```
ar5_3D.ndim
```

Out[41]:
```
3
```

In [42]:
```
ar5_3D.dtype
```

Out[42]:
```
dtype('int32')
```

In [43]:
```
ar5_3D.nbytes
```

Out[43]:
```
80
```

Accessing elements

```
Access elements using positive indexes . Positive Index starts from 0
Negative index starts from -1
array[row,column]
slicing : array[start:Stop:Step]
```

```
Positive index : Default value for start index : 0 and default value for
the stop index is the last element
```

In [44]:
```
ar1
```

Out[44]:
```
array([1, 2, 3, 4, 5])
```

In [45]:
```
ar1[0]
```

Out[45]:
```
1
```

In [46]:
```
ar1[1]
```

Out[46]:
```
2
```

In [47]:
```
ar1[-1]
```

Out[47]:
```
5
```

In [48]:
```
ar5_2D
```

Out[48]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [49]:
```
ar5_2D[2,2]
```

Out[49]:
```
11
```

In [50]:
```
ar5_2D[4,3]
```

Out[50]:
```
20
```

In [51]:
```
ar5_2D[2,1]
```

Out[51]:
```
10
```

In [52]:
```
ar1
```

Out[52]:
```
array([1, 2, 3, 4, 5])
```

In [53]:
```
ar1[1:4]
```

Out[53]:
```
array([2, 3, 4])
```

In [54]:
```
ar5_2D
```

Out[54]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [55]:
```
ar5_2D[1,1:3]
```

Out[55]:
```
array([6, 7])
```

In [56]:
```
ar1
```

Out[56]:
```
array([1, 2, 3, 4, 5])
```

In [57]:
```
ar1[:]
```

Out[57]:
```
array([1, 2, 3, 4, 5])
```

In [58]:
```
ar5_2D[:,1]
```

Out[58]:
```
array([ 2,  6, 10, 14, 18])
```

In [59]:
```
ar1
```

Out[59]:
```
array([1, 2, 3, 4, 5])
```

In [60]:
```
ar1[::3]
```

Out[60]:
```
array([1, 4])
```

In [61]:
```
ar5_2D
```

Out[61]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [62]:
```
ar5_2D[:,0::3]
```

Out[62]:
```
array([[ 1,  4],
       [ 5,  8],
       [ 9, 12],
       [13, 16],
       [17, 20]])
```

In [63]:
```
ar5_2D[:,3]
```

Out[63]:
```
array([ 4,  8, 12, 16, 20])
```

In [64]:
```
ar5_2D
```

Out[64]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [65]:
```
# Last row
ar5_2D[4::,::]
```

Out[65]:
```
array([[17, 18, 19, 20]])
```

In [66]:
```
# 10,11,14,15
ar5_2D[2:4,1:3]
```

Out[66]:
```
array([[10, 11],
       [14, 15]])
```

In [67]:
```
#10,11,12,13,14,15
ar5_2D[2:4,1:4]
```

Out[67]:
```
array([[10, 11, 12],
       [14, 15, 16]])
```

Update

In [68]:
```
ar1
```

Out[68]:
```
array([1, 2, 3, 4, 5])
```

In [69]:
```
ar1[1]=21
```

In [70]:
```
ar1
```

Out[70]:
```
array([ 1, 21,  3,  4,  5])
```

Mathematical operations

In [71]:
```
ar1
```

Out[71]:
```
array([ 1, 21,  3,  4,  5])
```

In [72]:
```
ar1+5
```

Out[72]:
```
array([ 6, 26,  8,  9, 10])
```

In [73]:
```
ar1-5
```

Out[73]:
```
array([-4, 16, -2, -1,  0])
```

In [74]:
```
ar1*3
```

Out[74]:
```
array([ 3, 63,  9, 12, 15])
```

In [75]:
```
ar1
```

Out[75]:
```
array([ 1, 21,  3,  4,  5])
```

In [76]:
```
ar2
```

Out[76]:
```
array([1, 2, 3, 4, 5])
```

In [77]:
```
ar1+ar2
```

Out[77]:
```
array([ 2, 23,  6,  8, 10])
```

In [78]:
```
ar5_2D
```

Out[78]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [79]:
```
ar5_2D*2
```

Out[79]:
```
array([[ 2,  4,  6,  8],
       [10, 12, 14, 16],
       [18, 20, 22, 24],
       [26, 28, 30, 32],
       [34, 36, 38, 40]])
```

In [80]:
```
ar1
```

Out[80]:
```
array([ 1, 21,  3,  4,  5])
```

In [81]:
```
ar5_2D+ar1
```

```
---------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
Cell In[81], line 1
----> 1 ar5_2D+ar1

ValueError: operands could not be broadcast together with shapes (5,4)
(5,)
```

In [82]:
```
ar7=[1,2,3,4]
```

In [83]:
```
ar5_2D+ar7
```

Out[83]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16],
       [14, 16, 18, 20],
       [18, 20, 22, 24]])
```

In [84]:
```
ar1.sum()
```

Out[84]:
```
34
```

In [85]:
```
ar1
```

Out[85]:
```
array([ 1, 21,  3,  4,  5])
```

In [86]:
```
ar1.mean()
```

Out[86]:
```
6.8
```

In [87]:
```
ar1.min()
```

Out[87]:
```
1
```

In [88]:
```
ar1.max()
```

Out[88]:
```
21
```

In [89]:
```
ar5_2D
```

Out[89]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20]])
```

In [90]:
```
ar5_2D.sum()
```

Out[90]:
```
210
```

In [91]:
```
ar5_2D[:,-1].sum()
```

Out[91]:
```
60
```

In [92]:
```
ar5_2D[0,:].mean()
```

Out[92]:
```
2.5
```

In [93]:
```
ar5_2D[ar5_2D>=10]
```

Out[93]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

In [94]:
```
ar5_2D[(ar5_2D>=10) & (ar5_2D<=15)]
```

Out[94]:
```
array([10, 11, 12, 13, 14, 15])
```

In [95]:
```
ar5_2D[ar5_2D%2!=0]
```

Out[95]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

Merging arrays

In [96]:
```
ar1
```

Out[96]:
```
array([ 1, 21,  3,  4,  5])
```

In [97]:
```
ar2
```

Out[97]:
```
array([1, 2, 3, 4, 5])
```

In [98]:
```
np.concatenate((ar1,ar2))
```

Out[98]:
```
array([ 1, 21,  3,  4,  5,  1,  2,  3,  4,  5])
```

In [99]:
```
np.concatenate((ar1,ar2))
```

Out[99]:
```
array([ 1, 21,  3,  4,  5,  1,  2,  3,  4,  5])
```

In [100]:
```
np.vstack((ar1,ar2))
```

Out[100]:
```
array([[ 1, 21,  3,  4,  5],
       [ 1,  2,  3,  4,  5]])
```

In [101]:
```
np.hstack((ar1,ar2))
```

Out[101]:
```
array([ 1, 21,  3,  4,  5,  1,  2,  3,  4,  5])
```

In [102]:
```
np.full((3,3,),10)
```

Out[102]:
```
array([[10, 10, 10],
       [10, 10, 10],
       [10, 10, 10]])
```

In [103]:
```
np.zeros((4,3),dtype=int)
```

Out[103]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [104]:
```
np.ones((4,3))
```

Out[104]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [105]:
```
ar10=np.empty((4,4))
```

In [106]:
```
ar10.fill(20)
```

In [107]:
```
ar10
```

Out[107]:
```
array([[20., 20., 20., 20.],
       [20., 20., 20., 20.],
       [20., 20., 20., 20.],
       [20., 20., 20., 20.]])
```

In [108]:
```
ar1
```

Out[108]:
```
array([ 1, 21,  3,  4,  5])
```

In [109]:
```
# Reversing an array
ar1[::-1]
```

Out[109]:
```
array([ 5,  4,  3, 21,  1])
```

In [110]:
```
np.flip(ar1)
```

Out[110]:
```
array([ 5,  4,  3, 21,  1])
```

In [111]:
```
# copy method and view method
ar10=np.copy(ar1)
```

In [112]:
```
ar11=ar1.view()
```

In [113]:
```
ar1
```

Out[113]:
```
array([ 1, 21,  3,  4,  5])
```

In [114]:
```
ar10
```

Out[114]:
```
array([ 1, 21,  3,  4,  5])
```

In [115]:
```
ar11
```

Out[115]:
```
array([ 1, 21,  3,  4,  5])
```

In [116]:
```
ar1[1]=15
```

In [117]:
```
ar1
```

Out[117]:
```
array([ 1, 15,  3,  4,  5])
```

In [118]:
```
ar10
```

Out[118]:
```
array([ 1, 21,  3,  4,  5])
```

In [119]:
```
ar11
```

Out[119]:
```
array([ 1, 15,  3,  4,  5])
```

```
First, the list of weights must be imported into Numpy as an array after
being converted from a list. After that, change all of the weights to be
in pounds by converting them from kilograms. For the purpose of making
your conversion, use the scalar conversion of 2.2 pounds per kilogram.
Finally, print the resultant array of weights using the pound
measurement.
Use following script for reference
weight_kg = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]
import numpy as np
# Create a numpy array np_weight_kg from weight_kg
# Create np_weight_lbs from np_weight_kg
# Print out np_weight_lbs
```

In [120]:
```
# Given list of weights in kilograms
weight_kg = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]
# Import Numpy
import numpy as np
# Create a numpy array np_weight_kg from weight_kg
np_weight_kg = np.array(weight_kg)
# Create np_weight_lbs from np_weight_kg by converting from kilograms to po
np_weight_lbs = np_weight_kg * 2.2
# Print out np_weight_lbs
print(np_weight_lbs)
```

```
[179.63  214.544 209.55  204.556 189.596 194.59 ]
```

```
Write a NumPy program to reverse an array (first element becomes last).
Sample Output
Original array:
[12 13 14 15 16 17 18 19 20 ]
                                                    Reverse array:
```

```
[20 19 18 17 16 15 14 13 12 ]
```

In [121]:
```python
#method 1
import numpy as np
# Original array
original_array = np.array([12, 13, 14, 15, 16, 17, 18, 19, 20])
# Reverse array using slicing with step -1
reverse_array = original_array[::-1]
# Print the original and reversed arrays
print("Original array:")
print(original_array)
print("Reverse array:")
print(reverse_array)
```

```
Original array:
[12 13 14 15 16 17 18 19 20]
Reverse array:
[20 19 18 17 16 15 14 13 12]
```

In [122]:
```python
#method 2
import numpy as np
# Original array
original_array = np.array([12, 13, 14, 15, 16, 17, 18, 19, 20])
# Reverse array using np.flip()
reverse_array = np.flip(original_array)
# Print the original and reversed arrays
print("Original array:")
print(original_array)
print("Reverse array:")
print(reverse_array)
```

```
Original array:
[12 13 14 15 16 17 18 19 20]
Reverse array:
[20 19 18 17 16 15 14 13 12]
```

In [ ]: