Creating NumPy Arrays

In [1]:
```
import numpy as np
```

In [2]:
```
a1 = np.arange(10)          # creates a range from 0 to 9
print(a1)                   # [0 1 2 3 4 5 6 7 8 9]
print(a1.shape)             # (10,)
```

```
[0 1 2 3 4 5 6 7 8 9]
(10,)
```

In [2]:
```
a2 = np.arange(0,10,4)      # creates a range from 0 to 9, step 2
print(a2)                   # [0 2 4 6 8]
```

```
[0 4 8]
```

In [3]:
```
a3 = np.zeros(5)            # create an array with all 0s
print(a3)                   # [ 0.  0.  0.  0.  0.]
print(a3.shape)             # (5,)
print(a3.ndim)
```

```
[0. 0. 0. 0. 0.]
(5,)
1
```

In [5]:
```
a4 = np.zeros((2,3))        # array of rank 2 with all 0s; 2 rows and 3 colum
print(a4.shape)             # (2,3)
print(a4)
```

```
(2, 3)
[[0. 0. 0.]
 [0. 0. 0.]]
```

In [6]:
```
a5 = np.full((2,3), 8)      # array of rank 2 with all 8s
print(a5)
```

```
[[8 8 8]
 [8 8 8]]
```

In [5]:
```
a6 = np.eye(4)              # 4x4 identity matrix
print(a6)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [6]:
```
np.eye?
```

In [8]:
```
a7 = np.random.random((2,4)) # rank 2 array (2 rows 4 columns) with random
                             # in the half-open interval [0.0, 1.0)
print(a7)
```

```
[[0.46733932 0.36103158 0.20873455 0.7589352 ]
 [0.50520187 0.03105929 0.60004448 0.09143153]]
```

In [11]:
```
list1 = [1,2,3,4,5]  # list1 is a list in Python
r1 = np.array(list1) # rank 1 array
print(r1)            # [1 2 3 4 5]
print(r1.shape)
```

```
[1 2 3 4 5]
(5,)
```

Array Indexing

In [10]:
```
print(r1[0])         # 1
print(r1[1])         # 2

print(r1[-1])        # 5
print(r1[-2])        # 4
```

```
1
2
5
4
```

In [13]:
```
list2 = [6,7,8,9,0]
r2 = np.array([list1,list2])  # rank 2 array
print(r2)
print(r2.shape)               # (2,5) - 2 rows and 5 columns
print(r2[0,0])                # 1
print(r2[0,1])                # 2
print(r2[1,0])                # 6
print(r2.ndim)
```

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

In [12]:
```
list1 = [1,2,3,4,5]
r1 = np.array(list1)
print(r1[[2,4]])              # [3 5]
```

```
[3 5]
```

Boolean Indexing

In [13]:
```
print(r1>2)     # [False False  True  True  True]
```

```
[False False  True  True  True]
```

In [14]:
```
print(r1[r1>2])    # [3 4 5]
```

```
[3 4 5]
```

Exercises

1. Print out all the odd number items in the r1 array
2. Print out the last third number in the r1 array

Solutions

In [15]:
```
print(r1[r1 % 2 == 1])

print(r1[-3])
```

```
[1 3 5]
3
```

In [16]:
```
nums = np.arange(20)
print(nums)        # [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

In [17]:
```
odd_num = nums[nums % 2 == 1]
print(odd_num)     # [ 1  3  5  7  9 11 13 15 17 19]
```

```
[ 1  3  5  7  9 11 13 15 17 19]
```

Slicing Arrays

In [18]:
```
a = np.array([[1,2,3,4,5],
              [4,5,6,7,8],
              [9,8,7,6,5]])    # rank 2 array
print(a)
```

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

In [19]:
```
b1 = a[1:3, :3]                # row 1 to 3 (not inclusive) and first 3 co
print(b1)
```

```
[[4 5 6]
 [9 8 7]]
```

In [20]:
```
b2 = a[-2:,-2:]
print(b2)
```

```
[[7 8]
 [6 5]]
```

NumPy Slice Is a Reference

In [21]:
```
b3 = a[1:, 2:]      # row 1 onwards and column 2 onwards
                    # b3 is now pointing to a subset of a7
print(b3)
```

```
[[6 7 8]
 [7 6 5]]
```

In [22]:
```
b3[0,2] = 88         # b3[0,2] is pointing to a[1,4]; modifying it will moc
                     # the original array
print(a)
```

```
[[ 1  2  3  4  5]
 [ 4  5  6  7 88]
 [ 9  8  7  6  5]]
```

In [23]:
```
b4 = a[2:, :]          # row 2 onwards and all columns
print(b4)              # b4 is rank 2
print(b4.shape)
```

```
[[9 8 7 6 5]]
(1, 5)
```

In [24]:
```
b5 = a[2, :]           # row 2 and all columns
print(b5)              # b5 is rank 1
```

```
[9 8 7 6 5]
```

In [25]:
```
print(b5.shape)        # (5,)
```

```
(5,)
```

Reshaping Arrays

In [26]:
```
b5 = b5.reshape(1,-1)
print(b5)
```

```
[[9 8 7 6 5]]
```

In [27]:
```
b4.reshape(-1,)
```

Out[27]:
```
array([9, 8, 7, 6, 5])
```

Array Maths

In [28]:
```
x1 = np.array([[1,2,3],[4,5,6]])
y1 = np.array([[7,8,9],[2,3,4]])

print(x1 + y1)
```

```
[[ 8 10 12]
 [ 6  8 10]]
```

In [29]:
```
x = np.array([2,3])
y = np.array([4,2])
z = x + y
```

In [30]:
```
np.add(x1,y1)
```

Out[30]:
```
array([[ 8, 10, 12],
       [ 6,  8, 10]])
```

In [31]:
```
print(x1 - y1)     # same as np.subtract(x1,y1)
print(x1 * y1)     # same as np.multiply(x1,y1)
print(x1 / y1)     # same as np.divide(x1,y1)
```

```
[[-6 -6 -6]
 [ 2  2  2]]
[[ 7 16 27]
 [ 8 15 24]]
[[0.14285714 0.25       0.33333333]
 [2.         1.66666667 1.5       ]]
```

In [32]:
```
names   = np.array(['Ann','Joe','Mark'])
heights = np.array([1.5, 1.78, 1.6])
weights = np.array([65, 46, 59])

bmi = weights/heights **2            # calculate the BMI
print(bmi)                           # [ 28.88888889  14.51836889  23.04687
```

```
[28.88888889 14.51836889 23.046875  ]
```

In [33]:
```
print("Overweight: "  , names[bmi>25])                    # Overweight:  ['
print("Underweight: " , names[bmi<18.5])                   # Underweight:  [
print("Healthy: "     , names[(bmi>=18.5) & (bmi<=25)])   # Healthy:  ['Mar
```

```
Overweight:  ['Ann']
Underweight:  ['Joe']
Healthy:  ['Mark']
```

Dot Product

In [34]:
```
x = np.array([2,3])
y = np.array([4,2])
np.dot(x,y)  # 2x4 + 3x2 = 14
```

Out[34]:
```
14
```

In [35]:
```
x2 = np.array([[1,2,3],[4,5,6]])
y2 = np.array([[7,8],[9,10], [11,12]])
print(np.dot(x2,y2))                    # matrix multiplication
```

```
[[ 58  64]
 [139 154]]
```

Matrix

In [36]:
```
x2 = np.matrix([[1,2],[4,5]])
y2 = np.matrix([[7,8],[2,3]])
```

In [37]:
```
x1 = np.array([[1,2],[4,5]])
y1 = np.array([[7,8],[2,3]])
x1 = np.asmatrix(x1)
y1 = np.asmatrix(y1)
```

In [38]:
```
x1 = np.array([[1,2],[4,5]])
y1 = np.array([[7,8],[2,3]])
print(x1 * y1)     # element-by-element multiplication

x2 = np.matrix([[1,2],[4,5]])
y2 = np.matrix([[7,8],[2,3]])
print(x2 * y2)    # dot product; same as np.dot()
```

```
[[ 7 16]
 [ 8 15]]
[[11 14]
 [38 47]]
```

Cumulative Sum

In [39]:
```
a = np.array([(1,2,3),(4,5,6), (7,8,9)])
print(a)
```

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

In [40]:
```
print(a.cumsum())   # prints the cumulative sum of all the
                    # elements in the array
                    # [ 1  3  6 10 15 21 28 36 45]
```

```
[ 1  3  6 10 15 21 28 36 45]
```

In [41]:
```
print(a.cumsum(axis=0))  # sum over rows for each of the 3 columns
```

```
[[ 1  2  3]
 [ 5  7  9]
 [12 15 18]]
```

In [42]:
```
print(a.cumsum(axis=1))  # sum over columns for each of the 3 rows
```

```
[[ 1  3  6]
 [ 4  9 15]
 [ 7 15 24]]
```

NumPy Sorting

In [43]:
```
ages = np.array([34,12,37,5,13])
sorted_ages = np.sort(ages)   # does not modify the original array
print(sorted_ages)            # [ 5 12 13 34 37]
print(ages)                   # [34 12 37  5 13]
```

```
[ 5 12 13 34 37]
[34 12 37  5 13]
```

In [44]:
```
ages.sort()                   # modifies the array
print(ages)                   # [ 5 12 13 34 37]
```

```
[ 5 12 13 34 37]
```

In [45]:
```
ages = np.array([34,12,37,5,13])
print(ages.argsort())         # [3 1 4 0 2]
```

```
[3 1 4 0 2]
```

In [46]:
```
print(ages[ages.argsort()])   # [ 5 12 13 34 37]
```

```
[ 5 12 13 34 37]
```

In [47]:
```
persons = np.array(['Johnny','Mary','Peter','Will','Joe'])
ages    = np.array([34,12,37,5,13])
heights = np.array([1.76,1.2,1.68,0.5,1.25])
```

In [48]:
```
sort_indices = np.argsort(ages)  # performs a sort based on ages
                                 # and returns an array of indices
                                 # indicating the sort order
```

In [49]:
```
print(persons[sort_indices])      # ['Will' 'Mary' 'Joe' 'Johnny' 'Peter']
print(ages[sort_indices])         # [ 5 12 13 34 37]
print(heights[sort_indices])      # [ 0.5   1.2   1.25  1.76  1.68]
```

```
['Will' 'Mary' 'Joe' 'Johnny' 'Peter']
[ 5 12 13 34 37]
[0.5  1.2  1.25 1.76 1.68]
```

In [50]:
```
sort_indices = np.argsort(persons)   # sort based on names
print(persons[sort_indices])         # ['Joe' 'Johnny' 'Mary' 'Peter' 'Will
print(ages[sort_indices])            # [13 34 12 37  5]
print(heights[sort_indices])         # [ 1.25  1.76  1.2   1.68  0.5 ]
```

```
['Joe' 'Johnny' 'Mary' 'Peter' 'Will']
[13 34 12 37  5]
[1.25 1.76 1.2  1.68 0.5 ]
```

In [51]:
```
reverse_sort_indices = np.argsort(persons)[::-1] # reverse the order of a l
print(persons[reverse_sort_indices])     # ['Will' 'Peter' 'Mary' 'Johnny'
print(ages[reverse_sort_indices])        # [ 5 37 12 34 13]
print(heights[reverse_sort_indices])     # [ 0.5   1.68  1.2   1.76  1.25]
```

```
['Will' 'Peter' 'Mary' 'Johnny' 'Joe']
[ 5 37 12 34 13]
[0.5  1.68 1.2  1.76 1.25]
```

Array Assignment

Copying by Reference

In [52]:
```
list1 = [[1,2,3,4], [5,6,7,8]]
a1 = np.array(list1)
print(a1)
```

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

In [53]:
```
a2 = a1              # creates a copy by reference
print(a1)
print(a2)
```

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

In [54]:
```
a2[0][0] = 11      # make some changes to a2
print(a1)          # affects a1
print(a2)
```

```
[[11  2  3  4]
 [ 5  6  7  8]]
[[11  2  3  4]
 [ 5  6  7  8]]
```

In [55]:
```
a1.shape = 1,-1   # reshape a1
print(a1)
print(a2)         # a2 also changes shape
```

```
[[11  2  3  4  5  6  7  8]]
[[11  2  3  4  5  6  7  8]]
```

Copying by View (Shallow Copy)

In [56]:
```
list1 = [[1,2,3,4], [5,6,7,8]]
a1 = np.array(list1)
a2 = a1.view()    # creates a copy of a1 by reference; but changes
                  # in dimension in a1 will not affect a2
print(a1)
print(a2)
```

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

In [57]:
```
a1[0][0] = 11     # make some changes in a1
print(a1)
print(a2)         # changes is also seen in a2
```

```
[[11  2  3  4]
 [ 5  6  7  8]]
[[11  2  3  4]
 [ 5  6  7  8]]
```

In [58]:
```
a1.shape = 1,-1   # change the shape of a1
print(a1)
print(a2)         # a2 does not change shape
```

```
[[11  2  3  4  5  6  7  8]]
[[11  2  3  4]
 [ 5  6  7  8]]
```

Copying by Value (Deep Copy)

In [59]:
```
list1 = [[1,2,3,4], [5,6,7,8]]
a1 = np.array(list1)
a2 = a1.copy()     # create a copy of a1 by value (deep copy)
```

In [60]:
```
a1[0][0] = 11      # make some changes in a1
print(a1)
print(a2)          # changes is not seen in a2
```

```
[[11  2  3  4]
 [ 5  6  7  8]]
[[1 2 3 4]
 [5 6 7 8]]
```

In [61]:
```
a1.shape = 1,-1    # change the shape of a1
print(a1)
print(a2)          # a2 does not change shape
```

```
[[11  2  3  4  5  6  7  8]]
[[1 2 3 4]
 [5 6 7 8]]
```

In [ ]:
```
```