In [1]:
```
# -*- coding: utf-8 -*-
```

In [2]:
```
import numpy as np
from timeit import default_timer as timer
```

1. Summing up 10M Numbers

In [3]:
```
a = list(range(10**7))
```

In [4]:
```
start = timer()
s = 0
for i in a:
    s += i
print(f"Time: {timer() - start} seconds")
```

```
Time: 0.645131874945946 seconds
```

In [5]:
```
start = timer()
s = sum(a)
print(f"Time: {timer() - start} seconds")
```

```
Time: 0.05586295807734132 seconds
```

In [6]:
```
x = np.array(a)
# x = np.arange(10**7)
x
```

Out[6]:
```
array([      0,       1,       2, ..., 9999997, 9999998, 9999999])
```

In [7]:
```
start = timer()
s = np.sum(x)
print(f"Time: {timer() - start} seconds")
```

```
Time: 0.009105166071094573 seconds
```

2. NumPy arrays: Vectors and Matrices

In [8]:
```
a = np.array([0,1,2])
a
```

Out[8]:
```
array([0, 1, 2])
```

In [9]:
```
a.shape
```

Out[9]:
```
(3,)
```

In [10]:
```
A = np.array([[0,1,2],[3,4,5]])
A
```

Out[10]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
In [11]: A.shape
```

```
Out[11]: (2, 3)
```

3. NumPy arrays != Lists

```
In [12]: a = list(range(4))
         a
```

```
Out[12]: [0, 1, 2, 3]
```

```
In [13]: a * 2 # * replicates
```

```
Out[13]: [0, 1, 2, 3, 0, 1, 2, 3]
```

```
In [14]: a + a # + concatenates
```

```
Out[14]: [0, 1, 2, 3, 0, 1, 2, 3]
```

```
In [15]: a.append(10)
         a
```

```
Out[15]: [0, 1, 2, 3, 10]
```

```
In [16]: a = np.arange(4)
         a
```

```
Out[16]: array([0, 1, 2, 3])
```

```
In [17]: a * 2 # * element-wise multiplies
```

```
Out[17]: array([0, 2, 4, 6])
```

```
In [18]: a + a # * element-wise sum
```

```
Out[18]: array([0, 2, 4, 6])
```

```
In [19]: a = np.append(a, 10)
         a
```

```
Out[19]: array([ 0,  1,  2,  3, 10])
```

5. Numpy array indexing

A list of lists

```
In [20]: A = [[0,1,2], [3,4,5]]
         A

Out[20]: [[0, 1, 2], [3, 4, 5]]
```

```
In [21]: A[0][0] # Return 1st row, 1st column

Out[21]: 0
```

```
In [22]: A[0] # 1st row

Out[22]: [0, 1, 2]
```

```
In [23]: [item[0] for item in A] # 1st column: Too complicated!

Out[23]: [0, 3]
```

```
In [24]: first_column = []
         # Iterate through the rows in A
         for row in A:
             # Append the first element (column) of each row to the first_column list
             first_column.append(row[0])
         first_column

Out[24]: [0, 3]
```

A Numpy matrix

```
In [25]: A = np.array([[0,1,2], [3,4,5]])
         A

Out[25]: array([[0, 1, 2],
                [3, 4, 5]])
```

```
In [26]: A[0,0] # 1st row, 1st column

Out[26]: 0
```

```
In [27]: A[0,:] # 1st row

Out[27]: array([0, 1, 2])
```

```
In [28]: A[:,0] # 1st column

Out[28]: array([0, 3])
```

6. Slice notation

```
In [29]: x = np.arange(10)
         x

Out[29]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [30]: x[0:2] # SAme as x[[0,1]]

Out[30]: array([0, 1])
```

```
In [31]: x[[0,1]]

Out[31]: array([0, 1])
```

```
In [32]: x[range(2)]

Out[32]: array([0, 1])
```

```
In [33]: x[1:] # Same as x[range(1, len(x))]

Out[33]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [34]: x[range(1, len(x))]

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

```
In [35]: x[:5]

Out[35]: array([0, 1, 2, 3, 4])
```

```
In [36]: x[range(5)]

Out[36]: array([0, 1, 2, 3, 4])
```

```
In [37]: x[:]

Out[37]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [38]: x[range(len(x))]

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

```
In [39]: x[:0:-1]

Out[39]: array([9, 8, 7, 6, 5, 4, 3, 2, 1])
```

```
In [40]: x[::-1]

Out[40]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

7. Boolean indexing

A Numpy vector

```
In [41]: x = np.array([0,3,1,4,2,5])
         x > 2
```

Out[41]:
```
array([False,  True, False,  True, False,  True])
```

In [42]:
```
x[x > 2]
```

Out[42]:
```
array([3, 4, 5])
```

A list

In [43]:
```
x = [0,3,1,4,2,5]
# x[x > 2]
```

In [44]:
```
x = [item for item in x if item > 2]
x
```

Out[44]:
```
[3, 4, 5]
```

A Numpy Matrix

In [45]:
```
A = np.array([[0,3,1], [4,2,5]])
A > 2
```

Out[45]:
```
array([[False,  True, False],
       [ True, False,  True]])
```

In [46]:
```
A[A > 2]
```

Out[46]:
```
array([3, 4, 5])
```

8. Copies vs Views

In [47]:
```
A = np.array([[0,1,2], [3,4,5]])
B = A
```

In [48]:
```
B[0,0] = 10
A[0,0]
```

Out[48]:
```
10
```

In [49]:
```
B = A.copy()
B[0,0] = 20
A[0,0]
```

Out[49]:
```
10
```

In [50]:
```
A[0,0]=1
row0 = A[0,:]
```

In [51]:
```
row0[0] = 5
```

In [52]:
```
A[0,0]
```

Out[52]:

```
5
```

In [53]:

```python
slice_copy = A[0,:].copy()
```

In [54]:

```python
slice_copy==A[0,:]
```

Out[54]:

```
array([ True,  True,  True])
```

9. Example: Gains and losses

In [55]:

```python
X = np.random.randint(-100,100, size=(10*365, 500))
```

For-loop

In [56]:

```python
start = timer()
gains = 0
losses = 0
for t in range(X.shape[0]):
    for j in range(X.shape[1]):
        if X[t,j] > 0:
            gains += X[t,j]
        else:
            losses += X[t,j]
print(f"Time: {timer() - start} seconds")
```

```
Time: 0.6244191670557484 seconds
```

No-loop

In [57]:

```python
start = timer()
gains = np.sum(X[X > 0])
losses = np.sum(X[X < 0])
print(f"Time: {timer() - start} seconds")
# The syntax print(f"") in Python refers to an f-string, which stands for "f
# Inside an f-string, you can include expressions inside {} braces,
# which will then be evaluated at runtime and then formatted using the provi
```

```
Time: 0.025519750081002712 seconds
```

10. Element-wise operators

In [58]:

```python
A = np.array([[3,2,1],[4,5,6]])
A
```

Out[58]:

```
array([[3, 2, 1],
       [4, 5, 6]])
```

In [59]:

```python
A * 2
```

Out[59]:

```
array([[ 6,  4,  2],
       [ 8, 10, 12]])
```

In [60]:

```python
A**2
```

```
Out[60]: array([[ 9,  4,  1],
                [16, 25, 36]])
```

```
In [61]: A * A
```

```
Out[61]: array([[ 9,  4,  1],
                [16, 25, 36]])
```

```
In [62]: np.dot(A, np.transpose(A)) # Matrix multiplication https://en.wikipedia.org/
```

```
Out[62]: array([[14, 28],
                [28, 77]])
```

Broadcasting

```
In [63]: x = np.array([0,1,2])
         A + x
```

```
Out[63]: array([[3, 3, 3],
                [4, 6, 8]])
```

11. Broadcasting

```
In [64]: x = np.array([0,10,20,30])
         x
```

```
Out[64]: array([ 0, 10, 20, 30])
```

```
In [65]: x.reshape((4,1))
```

```
Out[65]: array([[ 0],
                [10],
                [20],
                [30]])
```

```
In [66]: x.reshape((2,2))
```

```
Out[66]: array([[ 0, 10],
                [20, 30]])
```

```
In [67]: x = np.array([0,10,20,30])
         y = np.array([0,1,2])
         x[:, np.newaxis] + y
```

```
Out[67]: array([[ 0,  1,  2],
                [10, 11, 12],
                [20, 21, 22],
                [30, 31, 32]])
```

```
In [ ]:
```

12. Example: Total of outer product

```
In [68]: x = np.array([1,10,20,30])
         y = np.array([2,3,4,5])
         n = x.shape[0]
```

for-loops

```
In [69]: total = 0
         for i in range(n):
             for j in range(n):
                 total += x[i] * y[j]
         print(total)

         854
```

No loops!

```
In [70]: total = np.sum(x.reshape((n,1)) * y)
         print(total)

         854
```

Quiz: Broadcasting and reshape

```
In [71]: import numpy as np
         x = np.array([1, 2])
         x

Out[71]: array([1, 2])
```

```
In [72]: y = np.array([[3], [4]])
         y

Out[72]: array([[3],
                [4]])
```

```
In [73]: x+y

Out[73]: array([[4, 5],
                [5, 6]])
```

13. Aggregations and reductions

```
In [74]: import numpy as np
         A = np.array([[0,1,2],[3,4,5]])
         A

Out[74]: array([[0, 1, 2],
                [3, 4, 5]])
```

```
In [75]: np.min(A)

Out[75]: 0
```

```
In [76]: np.min(A,axis=0) #minimum of each column
```

Out[76]:

```
array([0, 1, 2])
```

In [77]:

```
np.min(A,axis=1) #minimum of each row
```

Out[77]:

```
array([0, 3])
```

Quiz 3: Maximum Mean Squared Error

In [78]:

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

Out[78]:

```
array([[0, 1, 2],
       [3, 4, 5]])
```

In [79]:

```
obs = np.array([3,1,1])
obs
```

Out[79]:

```
array([3, 1, 1])
```

In [80]:

```
np.mean((P-obs)**2,axis = 1)# axix = 1, along columns direction, operate on
```

Out[80]:

```
array([3.33333333, 8.33333333])
```

In [81]:

```
np.max(np.mean((P-obs)**2,axis = 1))
```

Out[81]:

```
8.333333333333334
```

14. Function vs. Methods

In [82]:

```
np.min(A)
```

Out[82]:

```
0
```

In [83]:

```
A.min()
```

Out[83]:

```
0
```

In [84]:

```
np.min(A,axis=0)
```

Out[84]:

```
array([0, 1, 2])
```

In [85]:

```
A.min(axis=0)
```

Out[85]:

```
array([0, 1, 2])
```

In [86]:

```
x = [1,2,3]
np.min(x)
```

Out[86]:

```
1
```

15. Boolean arrays: Any vs All

In [87]:
```
b = np.array([1, 1, 0, 0]) # 1 is True, 0 is False
```

In [88]:
```
np.logical_not(b)
```

Out[88]:
```
array([False, False,  True,  True])
```

In [89]:
```
np.logical_and(b, b)
```

Out[89]:
```
array([ True,  True, False, False])
```

In [90]:
```
np.logical_or(b, b)
```

Out[90]:
```
array([ True,  True, False, False])
```

In [91]:
```
np.all(b) # all TRUE?
```

Out[91]:
```
False
```

In [92]:
```
np.any(b) # any TRUE?
```

Out[92]:
```
True
```

In [93]:
```
B = np.array([[1,0,1],[0,1,1]])
np.all(B, axis = 0)
```

Out[93]:
```
array([False, False,  True])
```

In [94]:
```
np.any(B, axis = 1)
```

Out[94]:
```
array([ True,  True])
```

Quiz 4

In [95]:
```
A = np.array([[-1, 2, 3],[-4, -5, 6]])
print(A)
```

```
[[-1  2  3]
 [-4 -5  6]]
```

In [96]:
```
np.any(A < 0)
```

Out[96]:
```
True
```

In [97]:
```
np.all(A < 0)
```

Out[97]:
```
False
```

In [98]:
```
np.all(A < 0, axis = 0)
```

```
Out[98]:  array([ True, False, False])

In [99]:  np.all(A < 0, axis = 1)

Out[99]:  array([False, False])
```

16. Sorting

```
In [100…  x = np.random.randint(0,100, size=10)

In [101…  np.sort(x)

Out[101]:  array([16, 28, 64, 78, 82, 82, 83, 90, 91, 92])

In [102…  -np.sort(-x)

Out[102]:  array([92, 91, 90, 83, 82, 82, 78, 64, 28, 16])

In [103…  A

Out[103]:  array([[-1,  2,  3],
                  [-4, -5,  6]])

In [122…  np.sort(A, axis=0)

Out[122]:  array([[-1,  2,  3],
                  [-5, -4,  6]])

In [105…  x = np.array([11,12,10,9])

In [106…  np.sort(x)

Out[106]:  array([ 9, 10, 11, 12])

In [107…  -np.sort(-x)

Out[107]:  array([12, 11, 10,  9])
```

Indirect sorting

```
In [108…  x = np.array([12,11,10,9])
          x

Out[108]:  array([12, 11, 10,  9])

In [109…  np.sort(x)

Out[109]:  array([ 9, 10, 11, 12])

In [110…  np.argsort(x)
```

```
Out[110]: array([3, 2, 1, 0])
```

```
In [111… x[np.argsort(x)]
```

```
Out[111]: array([ 9, 10, 11, 12])
```

17. Example: Find the minimum by sorting

```
In [112… x = np.arange(0,6,0.0000001)
```

```
In [113… y = np.cos(x)
```

```
In [114… i = np.argmin(y)
```

```
In [115… print(x[i])
```

```
3.1415927
```