```
#import numpy library
import numpy as np

#make an array of the values
array = [[14695.80, 15279.00, 14307.00, 14656.20], [14681.90,
14681.90, 12350.10, 12952.20],
        [12897.70, 14377.40, 12755.60, 14156.40], [14112.20, 14112.20,
13154.70, 13657.20],
        [13625.00, 15444.60, 13163.60, 14982.10], [14978.20, 15572.80,
14844.50, 15201.00],
        [15270.70, 15739.70, 14522.20, 15599.20], [15477.20, 17705.20,
15202.80, 17429.50],
        [17462.10, 17712.40, 16764.60, 17527.00], [17527.30, 17579.60,
16087.70, 16477.60],
        [16476.20, 16537.90, 14208.20, 15170.10], [15123.70, 15497.50,
14424.00, 14595.40],
        [14588.50, 14973.30, 13691.20, 14973.30]]

#making a numpy array
my_array = np.array(array)
print(my_array)
```

```
[[14695.8 15279.  14307.  14656.2]
 [14681.9 14681.9 12350.1 12952.2]
 [12897.7 14377.4 12755.6 14156.4]
 [14112.2 14112.2 13154.7 13657.2]
 [13625.  15444.6 13163.6 14982.1]
 [14978.2 15572.8 14844.5 15201. ]
 [15270.7 15739.7 14522.2 15599.2]
 [15477.2 17705.2 15202.8 17429.5]
 [17462.1 17712.4 16764.6 17527. ]
 [17527.3 17579.6 16087.7 16477.6]
 [16476.2 16537.9 14208.2 15170.1]
 [15123.7 15497.5 14424.  14595.4]
 [14588.5 14973.3 13691.2 14973.3]]
```

Q1: Write the program using NUMPY arrays to calculate HLC/3 and SMA9

```
#here we are calculating hlc/3 which is actually mean of the last
three values in each row
hlc3_array = []
for i in range(len(my_array)):
    mean = np.mean(my_array[i, [1, 2, 3]])
    hlc3_array.append(mean)
print(hlc3_array)
```

```
[14747.4, 13328.066666666666, 13763.133333333333, 13641.366666666669,
14530.1, 15206.1, 15287.033333333335, 16779.166666666668,
```

```
17334.666666666668, 16714.966666666667, 15305.400000000001,
14838.966666666667, 14545.933333333334]

#now we are calculating SMA9

#access closing price
last_column = my_array[:, -1]

sma9_array = []
i = 0
for i in range(5):
    sma_mean = np.mean(last_column[i:i+9])
    sma9_array.append(sma_mean)
print(sma9_array)
```

```
[15128.977777777776, 15331.355555555558, 15577.788888888887,
15626.566666666668, 15772.799999999997]
```

2- We want to get all values from a numpy array that satisfy a certain condition, For example:

Na=[1,3,5,3,6,2,8,9,10] element less than 6 result =[1,3,5,3,2]

```
na = [1, 3, 5, 3, 6, 2, 8, 9, 10]

#convert into numpy array
new_array = np.array(na)

print(new_array<6)
print(new_array[new_array < 6])
```

```
[ True  True  True  True False  True False False False]
[1 3 5 3 2]
```

3-Remember the exercise used in theoretical classes with weights. How can we test the initial array to make sure that no array values' weights are zero? Of course without using cycle

```
#lets take that weight array from class exercise
weights= np.array([72, 35, 64, 88, 51, 90, 74, 12])

#we can use all() function and it will tell us false if there is zero

print(np.all(weights))
#print(np.any(weights))

#lets check by putting one zero in the array
```

```
zero_weights = np.array([72, 35, 64, 88, 51, 90, 74, 12, 0])
print(np.all(zero_weights))
#print(np.any(zero_weights))
```

```
True
False
```

so we can see that in our array there are no zero values

4-We want to know if 2 numpy arrays are more or less equal. More or less means that two by two the elements are different only by an epsilon tolerance. Is there a method ? Write the function that uses the method. What happens if some array values are unknown? N/A type of excel?

```
#first we can use eqiality operator to compare the arrays

array1 = np.array([[1, 2], [2, 1]])
array2 = np.array([[2, 3], [4, 5]])

compare = array1 == array2
equal_arrays = compare.all()
print(equal_arrays)

array3 = np.array([1, 2, 3, 4])
array4 = np.array([1, 2, 3, 4])

comp = array3 == array4
equ_arr = comp.all()
print(equ_arr)
```

```
False
True
```

As we ca see when array sare same we get true and when they are different we get false

```
#we can also do this comaparison by epilson tolerance using allclose
function
# we can use equal_nan is equal to true for the na values like below

arr1 = np.array([1.00000, 2.000, 0.00001])
arr2 = np.array([1.00000, 2.000, 0.0001])

equal = np.allclose(arr1, arr2)
print(equal)

equality = np.allclose([2.00, 3.00, np.nan], [2.00, 3.00, np.nan],
equal_nan=True)
```

```
print(equality)
```

```
False
True
```

5- The file used in exercise one is organized by date. That is, each line corresponds to a date. If you consider that the first line of the file has today's date and what follows are dates from previous days, write a function that reads the file and returns a dictionary that has the dates as key and the elements of each file line as data in the form of an array of numpy.

```
#reading again the original array
my_array = np.array(array)
print(my_array)
```

```
[[14695.8 15279.  14307.  14656.2]
 [14681.9 14681.9 12350.1 12952.2]
 [12897.7 14377.4 12755.6 14156.4]
 [14112.2 14112.2 13154.7 13657.2]
 [13625.  15444.6 13163.6 14982.1]
 [14978.2 15572.8 14844.5 15201. ]
 [15270.7 15739.7 14522.2 15599.2]
 [15477.2 17705.2 15202.8 17429.5]
 [17462.1 17712.4 16764.6 17527. ]
 [17527.3 17579.6 16087.7 16477.6]
 [16476.2 16537.9 14208.2 15170.1]
 [15123.7 15497.5 14424.  14595.4]
 [14588.5 14973.3 13691.2 14973.3]]
```

```
#import date and time
from datetime import datetime as dt
from datetime import timedelta as td
```

```
#empty dicationary, later we will add element into it
date_array ={}
```

```
#end and start dates
starting_date = '2022-09-03'
ending_date = '2022-09-15'
```

```
#format
start_date = dt.strptime(starting_date,'%Y-%m-%d')
```

```
end_date = dt.strptime(ending_date,'%Y-%m-%d')
delta = end_date - start_date
Num = len(my_array)-1


#using for loop to add into the dicationary
for i in range(delta.days+1):
    date_array[start_date + td(days=i)] = my_array[Num]
    Num-=1
print(date_data)
```

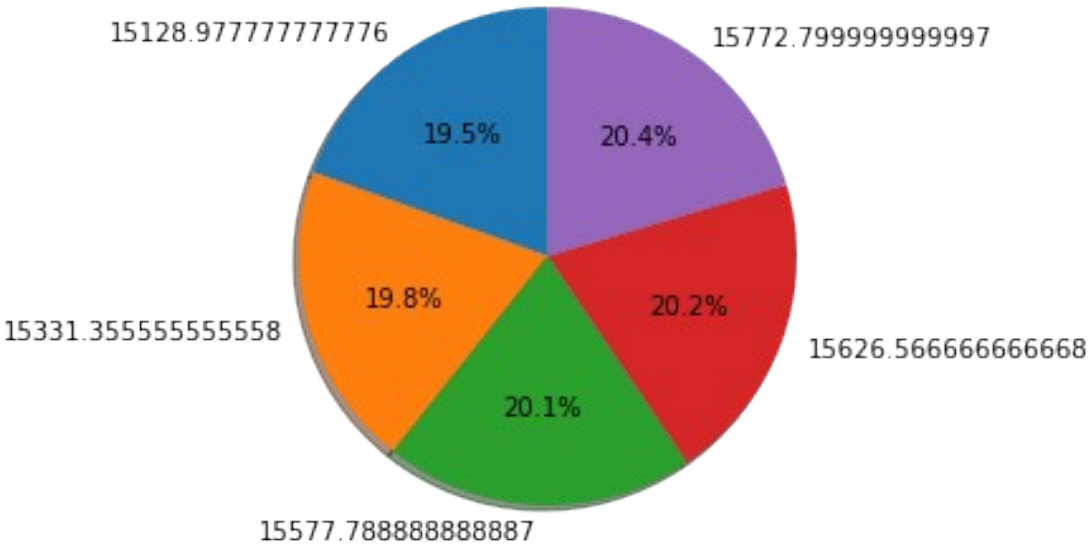
```
{datetime.datetime(2022, 9, 3, 0, 0): array([14588.5, 14973.3,
13691.2, 14973.3]), datetime.datetime(2022, 9, 4, 0, 0):
array([15123.7, 15497.5, 14424. , 14595.4]), datetime.datetime(2022,
9, 5, 0, 0): array([16476.2, 16537.9, 14208.2, 15170.1]),
datetime.datetime(2022, 9, 6, 0, 0): array([17527.3, 17579.6, 16087.7,
16477.6]), datetime.datetime(2022, 9, 7, 0, 0): array([17462.1,
17712.4, 16764.6, 17527. ]), datetime.datetime(2022, 9, 8, 0, 0):
array([15477.2, 17705.2, 15202.8, 17429.5]), datetime.datetime(2022,
9, 9, 0, 0): array([15270.7, 15739.7, 14522.2, 15599.2]),
datetime.datetime(2022, 9, 10, 0, 0): array([14978.2, 15572.8,
14844.5, 15201. ]), datetime.datetime(2022, 9, 11, 0, 0):
array([13625. , 15444.6, 13163.6, 14982.1]), datetime.datetime(2022,
9, 12, 0, 0): array([14112.2, 14112.2, 13154.7, 13657.2]),
datetime.datetime(2022, 9, 13, 0, 0): array([12897.7, 14377.4,
12755.6, 14156.4]), datetime.datetime(2022, 9, 14, 0, 0):
array([14681.9, 14681.9, 12350.1, 12952.2]), datetime.datetime(2022,
9, 15, 0, 0): array([14695.8, 15279. , 14307. , 14656.2])}
```

6- How about changing last week's exercise code to obtain a pie chart but with the SMA9 data obtained in exercise 1?

```
#sma9 array
import matplotlib.pyplot as plt
print(sma9_array)

#now make a pie chart using these vlaues
fig1,ax1=plt.subplots()
ax1.pie(sma9_array, labels = sma9_array, autopct='%1.1f%
%',shadow=True,startangle=90)
ax1.axis('equal')
plt.show()
```

```
[15128.977777777776, 15331.355555555558, 15577.788888888887,
15626.566666666668, 15772.799999999997]
```


15128.977777777776
15772.799999999997
19.5%
20.4%
19.8%
20.2%
15331.355555555558
15626.566666666668
20.1%
15577.788888888887