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

In [6]:
```
def compute_reciprocals_usingList(values):
    output=[]
    for i in range(len(values)):
        if(values[i]!=0):
            output.append(1.0/values[i])

        else:
            output[i]=None

    return output
```

In [13]:
```
def compute_reciprocals_usingPythonArray(values):
    output=array.array('f',[])
    for i in range(len(values)):
        if(values[i]!=0):
            output.append(1.0/values[i])

        else:
            output[i]=None

    return output
```

In [18]:
```
values_list=list(np.random.randint(1,10, size=1000000))
print(len(values_list))

values_array=array.array('i',list(np.random.randint(1,10, size=1000000)
print(len(values_array))
```

```
1000000
1000000
```

In [19]:
```
%timeit compute_reciprocals_usingList(values_list)
```

```
1.25 s ± 15.5 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [20]:
```
%timeit compute_reciprocals_usingPythonArray(values_array)
```

```
166 ms ± 1.65 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [21]:
```
#how much optimization is too much optimization?
```

In [22]:
```
big_array=np.random.randint(1,10, size=1000000)
len(big_array)
```

Out[22]:
```
1000000
```

In [23]:
```
%timeit list(map(lambda x: 1/x, list(big_array)))
```

```
173 ms ± 8.36 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [24]:
```
%timeit array.array('f',map(lambda x: 1/x, array.array('f',big_array))
```

```
189 ms ± 3.43 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [25]:
```
#improve using vectorised operations
```

In [26]:
```
%timeit (1.0/big_array)
```

```
597 µs ± 12.5 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops ea
ch)
```

In [27]:
```
#can we try improving it further - funtional + vectorised

%timeit np.array(map(lambda x: 1/x, big_array))
```

```
1.11 µs ± 4.89 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loo
ps each)
```

In [28]:
```
#In-memory
%timeit (map(lambda x: 1/x, big_array))
```

```
137 ns ± 1.46 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loo
ps each)
```

In [32]:
```
x=map(lambda x: 1/x, big_array)
```

Out[32]:
```
<map at 0x7fc9b91f60d0>
```

In [33]:
```
#WAP to reverse a string

s='suraaj'

s[::-1]
```

Out[33]:
```
'jaarus'
```

In [34]:
```
x=list(np.random.randint(1,10, size=15))
```

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

Out[35]:
```
[4, 8, 9, 2, 6, 8, 2, 3, 6, 3, 4, 8, 4, 5, 6]
```

In [36]:
```
#WAP to count the frequency of each element in the list

freq_dic={}

for i in x:
    if i in freq_dic:
        freq_dic[i]+=1
    else:
        freq_dic[i]=1
```

In [37]:
```
freq_dic
```

Out[37]:
```
{4: 3, 8: 3, 9: 1, 2: 2, 6: 3, 3: 2, 5: 1}
```

In [38]:
```
x=list(np.random.randint(1,10, size=15))
y=list(np.random.randint(1,10, size=15))
```

In [39]:
```
print(x)
print(y)
```

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

In [40]:
```
#WAP to find common elements between these 2 lists
common_list=[]

for i in x:
    if i in y:
        common_list.append(i)
```

In [41]:
```
common_list
```

Out[41]:
```
[9, 1, 1, 9, 9, 3, 9, 9, 3, 2]
```

In [42]:
```
#remove the duplicates without using set
unique_list=[]

for i in common_list:
    if i not in unique_list:
        unique_list.append(i)
```

In [43]:
```
unique_list
```

Out[43]:
```
[9, 1, 3, 2]
```

In [47]:
```
#WAP TO FIND THE SECOND LARGEST ELEMENT FROM THE LIST

def second_largest_picker(x_list):
    #reassignment

    largest= float('-inf')
    second_largest= float('-inf')

    for item in x_list:
        if item >largest:
            second_largest=largest
            largest=item  #reassignment

        if item> second_largest and item!=largest:
            second_largest=item

    return second_largest
```

In [48]:
```
second_largest_picker(unique_list)
```

Out[48]:
```
3
```

In [49]:
```
second_largest_picker(x)
```

Out[49]:
```
5
```

In [ ]:
```
#logs complexity
#space complexity
```

In [58]:
```
#recursion

def factorial(n):
    if n ==0:
        return 1

    else:
        return n*factorial(n-1)
```

In [59]:
```
factorial(5)
```

```
5 *  4
4 *  3
3 *  2
2 *  1
1 *  0
1
2
6
24
120
```

Out[59]:
```
120
```

In [ ]:
```

```