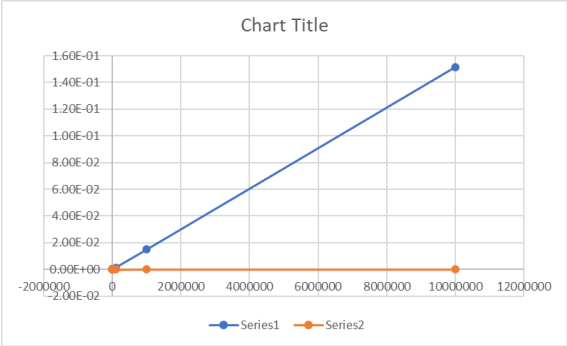
Set Class, Dictionary Class, and Map ADT

- We have learned that a set is unordered and allows no duplicates, so it sounds like we need to use a linear search in a set object (in Python) to find an element. But in fact, this is not true.

1. Search number 2×10^9 in a list and a set with n numbers. What are the time complexities of these two operations?



   - It turns out that the searching operation in a set with n elements only needs $O(1)$ time. How is this possible?

- The set class in Python is an implementation of the **Map (or Dictionary)** abstract data type.

❖ As an abstract data type, maps store paired data called keys and values. k : v
   - Each key in a map is unique, or in other words, no duplicate keys are allowed in a map.
   - For a certain key, its value should also be certain. But different keys might have the same value.

❖ Maps provide at least the following methods:
   - __setitem__(key, value): add a new key-value pair to the map. If the key already exists in the map, update its old value to the new one.
   - __getitem__(key): retrieve the paired value for the given key.
   - __contains__(key): returns whether the given key is in the map.

- From this big picture, we can see that a map is used to quickly retrieve one data once its pair data is provided.
- Note that, in Pyhton, the map ADT or the dictionary ADT is also implemented as the **dict** class. A dict is like a set of key-value pairs. For example, we can create the following dict in python:

$$\{"100": "apple", \quad 200: "banana", \quad "300": "cherry", \quad "400": "apple"\}$$

- A set is actually an implementation of Map ADT with all values omitted.
   - A set can be considered as a dict with all keys mapped to None. Using this idea, we can implement a set class using dict class.

$$\{3, 5, 2, 7\} \rightarrow \{3{:}\text{None}, \quad 5{:}\text{None}, \quad 2{:}\text{None}, \quad 7{:}\text{None}\}$$

A Naïve Implementation of the Map ADT

- From the above example, we know that if we have a full implementation of the Map ADT, we can create the set class. Now let's focus on how the Map ADT is implemented.

❖ The most naïve idea is to implement it with an 2D array.

K1	K2	K3	K4	K5					
V1	~~V2~~ V3	V1	V3	V3					

set(K5, V3) set(K2, V3)

- What is the time complexity of each Map method?
  - __setitem__(key, value): we need to scan the first row of the array to check whether the key is there: if no, we add this new pair to the array at the next available index; if yes, we update the old value to the new one. This operation takes $O(n)$ time.
  - __getitem__(key): we need to scan the array to search for the key, when we find it, we can return its paired value. This operation takes $O(n)$ time.
  - __contains__(key): similar to getitem, we need scan the array to search for the key. It takes $O(n)$ time.

  - All three most important methods need linear time, so this is not a good design.

Hash and Hashtable

- **Hash** is number that is calculated for an object. In python, there is the hash method implemented, it creates a random integer for an object.
  - The hash for an object will be temporarily stored. But when we re-interpret, an object will be given a new has.
  - Different object has different hash. When we have only a small number of objects, even if we only look at the last several digits, their hashes are ***most likely*** different.

- In the previous design for Map ADT, we store a key-value pair into the next available index in an array. Now, instead, if we store the pair at $index = hash(key)\ \%\ len(array)$, then the location of the pair is immediately known (within this job).
- A **hashtable** is an implementation of the Map that uses the hash for a key to compute an index into an array where the corresponding key-value pair will be stored.