**Time Complexity & Big ‘O’ notation**

If we take a algorithm or a function, how much time they take to complete the program?

But execution speed is dependent on many factors, like cpu, os, and cpu load.

So looking at the execution time of a program is not the best way to calculate time complexity.

It is wise to ask how does the runtime of the function grow with respect to size of input.

Time complexity is represented by big 'O' notation

**Big 'O' notation :**

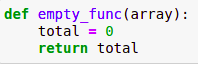
It is the easy way to represent time complexity by big 'O' notation

O(g(x)) where g(x) describes the behaviour of runtime as input size increases

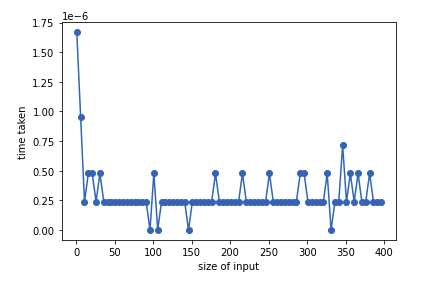
There are different type of function as given below

**Constant time O(1) :**

Execution time doesn't increase with respect to size of input



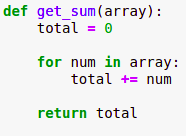
Above function doesn’t depends on the size of the input, So no matter the input size runtime will be constant time



It is the runtime of function with respect to size of input, runtime stays constant

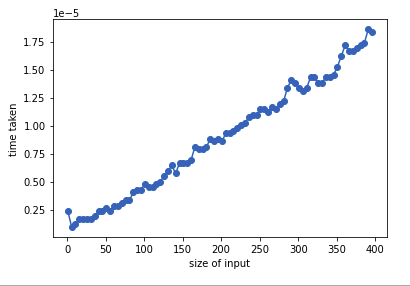
**Linear time O(n) :**

Runtime of the function or algorithm grows linearly with increase in size of input



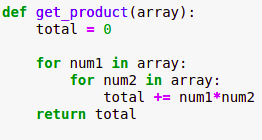
Above program takes a list as an input and calculates the sum of all elements in the list.

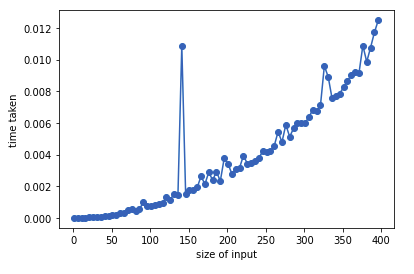
Below is the graph of runtime of above program as the size of input increases



**Quadratic time O(n^2):**

Runtime of a function looks like a function x^2





def get\_sum(array): #time taken = c1

total = 0 #c2

for num in array: #c3 \* (n+1)

total += num #c4 \* n

return total #c5

c1 + c2 + c5 + c3\*(n+1) + c4\*n

C1 + n\*c3 + c3 + n\*c4

C2 + n(c3+c4)

C2 + n\*C3 ==> O(n) linear time