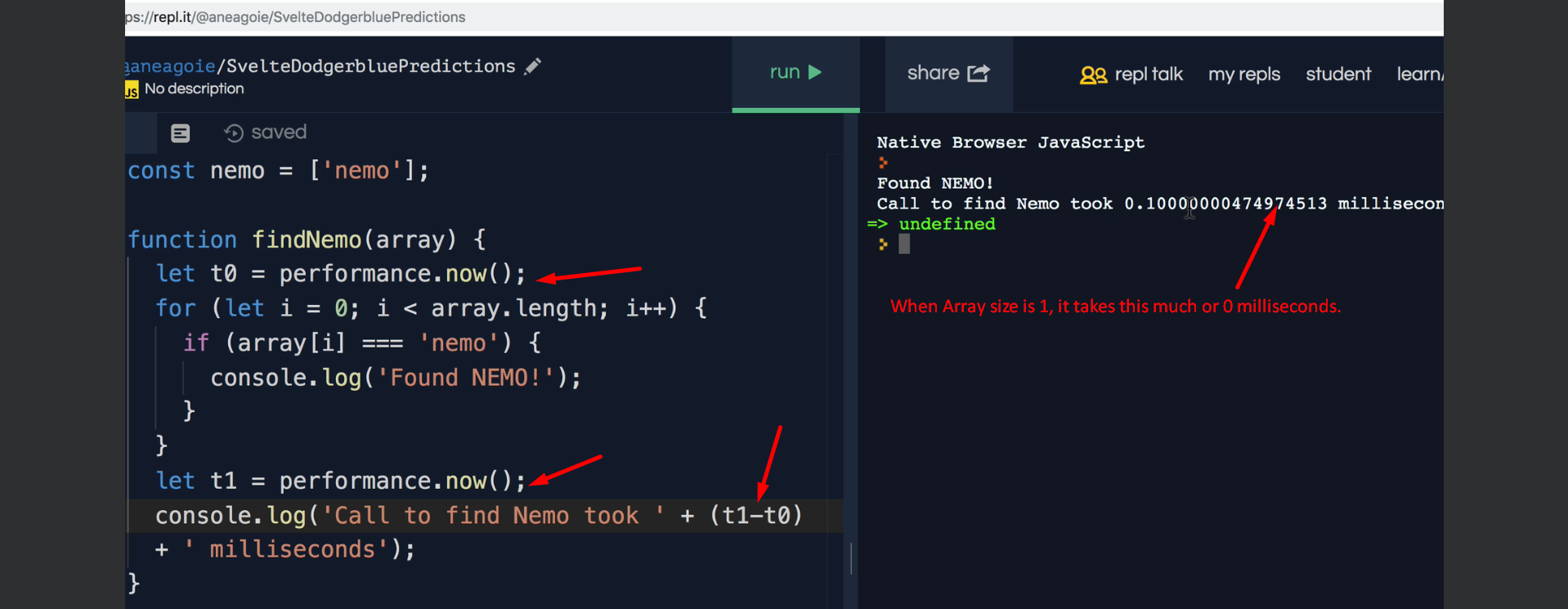
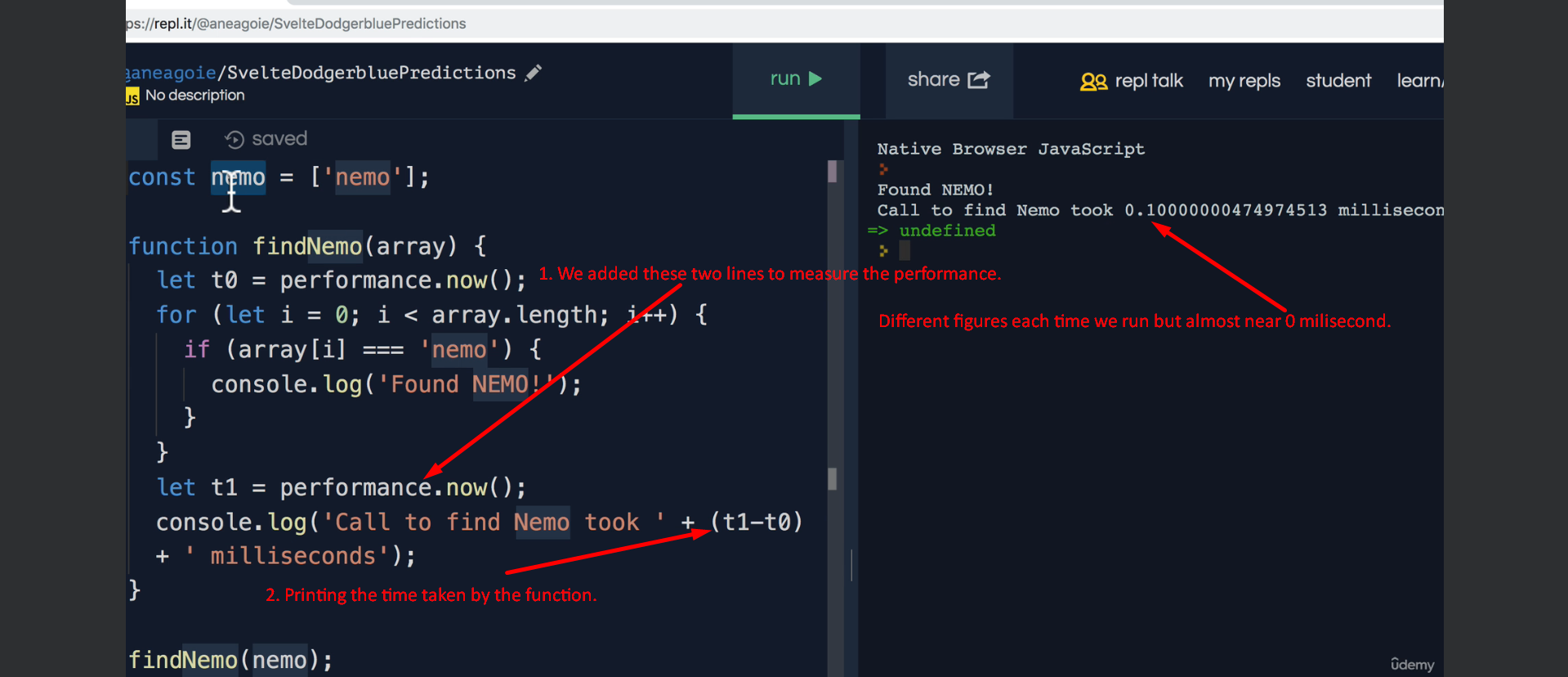
1. Let’s continue our discussion what this **scalability idea means**.
2. 
3. Note that we have changed the size of array from 1 to 10.
4. 
5. Even though instructor shown by running the code but I am taking a tabular for analysis purpose.

|  |  |
| --- | --- |
| Algo to check if a given String is there in String Array. | |
| Array Size | **Time Taken (Milliseconds)** |
| 1 | 0 or .10 |
| 100 | .5 |
| 1,000 | 7 |
| 10,000 | 46 |
| 1,00,000 | 343 = .34 seconds |

What did we observe here?  
We observed that as our input grows, our function **findNemo(Array)** becomes slower and slower.   
This observed analysis depends on many factors.

* 1. How powerful the CPU is.
  2. What other programs are running on our system.
  3. Programming Language.
  4. Many other factors.

1. Usually, we write code but we deploy on a different machine. So, we can measure the efficiency of a code by running it on some computer.
2. So, there are many questions.
   1. Good Code?
   2. Bad Code?
   3. How code will scale as the number of input increases and it doesn’t constantly slow down more and more.
3. Big O Notation is the Language we use for talking about how long an Algo will take to run regardless the computer differences.
4. When we talk about Big O and Scalability of a code, we simply mean when we grow bigger and bigger with input, how much the algorithm or function slows down.  
   For example: We the number of elements in the String Array increases, how many more operations do we have to do to search for “Nemo”?  
   This is what we call **Algorithm Efficiency**.
5. Remember: By Big O Notation and Scalability of Code, we simply mean when we grow bigger and bigger with our input, how much the Algorithm slows down.   
   The less it slows down or the lesser it slows down, the better it is.
6. Big O Notation allows us to figure out the number of steps an Algo will take.
7. Let’s go deeper in lecture.