Time Complexity

- We mentioned about the Zen of Python, where "Simple is better than complex." But how do you determine whether your code is "simple"?
- We can use time and space complexities to quantify the simplicity of your code.
We mentioned about the Zen of Python, where "Simple is better than complex." But how do you determine whether your code is "simple"?

We can use time and space complexities to quantify the simplicity of your code.

Time complexity refers to the amount of time an algorithm takes to run. But instead of using the physical time required, we use the number of operations relative to your input size to represent the time complexity.

We use the big $O$ notation, e.g. $O(n^2)$. 
For example, $O(1)$ refers to a constant time complexity. This means the runtime is always a constant, regardless of the input size. If your code requires exactly 3 operations, we will just call it $O(1)$.

If your input size is $n$, and you found your code requires $5n$ steps, we will just call the time complexity to be $O(n)$.

When talking about time complexity, we always keep the leading order term. For example, if your code requires $0.5n^2 + 3n$ steps, we will just call the time complexity to be $O(n^2)$.

In coding interviews, you will always be asked: What is the time complexity of your code? Can you write better code with lower time complexity?
Time Complexity

- In general, the following orders hold:

\[ O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^k) < O(2^n) < O(n!) \] (1)
For space complexity, we are talking about **memory usage**. It measures the amount of memory or storage space required by an algorithm to solve a problem in relation to the size of the input data.

It only refers to memory used for variables, data structures you used, and any additional storage required during the execution of the algorithm (not including the memory needed to store the input data).

In general, there can be a trade-off between time and space complexities. Although it depends on your specific problems at hand, it is generally a good idea to trade space complexity for lower time complexity in optimization (time complexity is more important!).
A Fibonacci sequence is

\[ 0, 1, 1, 2, 3, 5, 8, 13, \ldots \]  

Write a simple code that calculates the \( n \)-th Fibonacci number. What is the time complexity of your code? Can you calculate the 50-th Fibonacci number within a reasonable time?

Can you write a better code that reduces the time complexity? Think about storing the values of the calculated Fibonacci numbers.