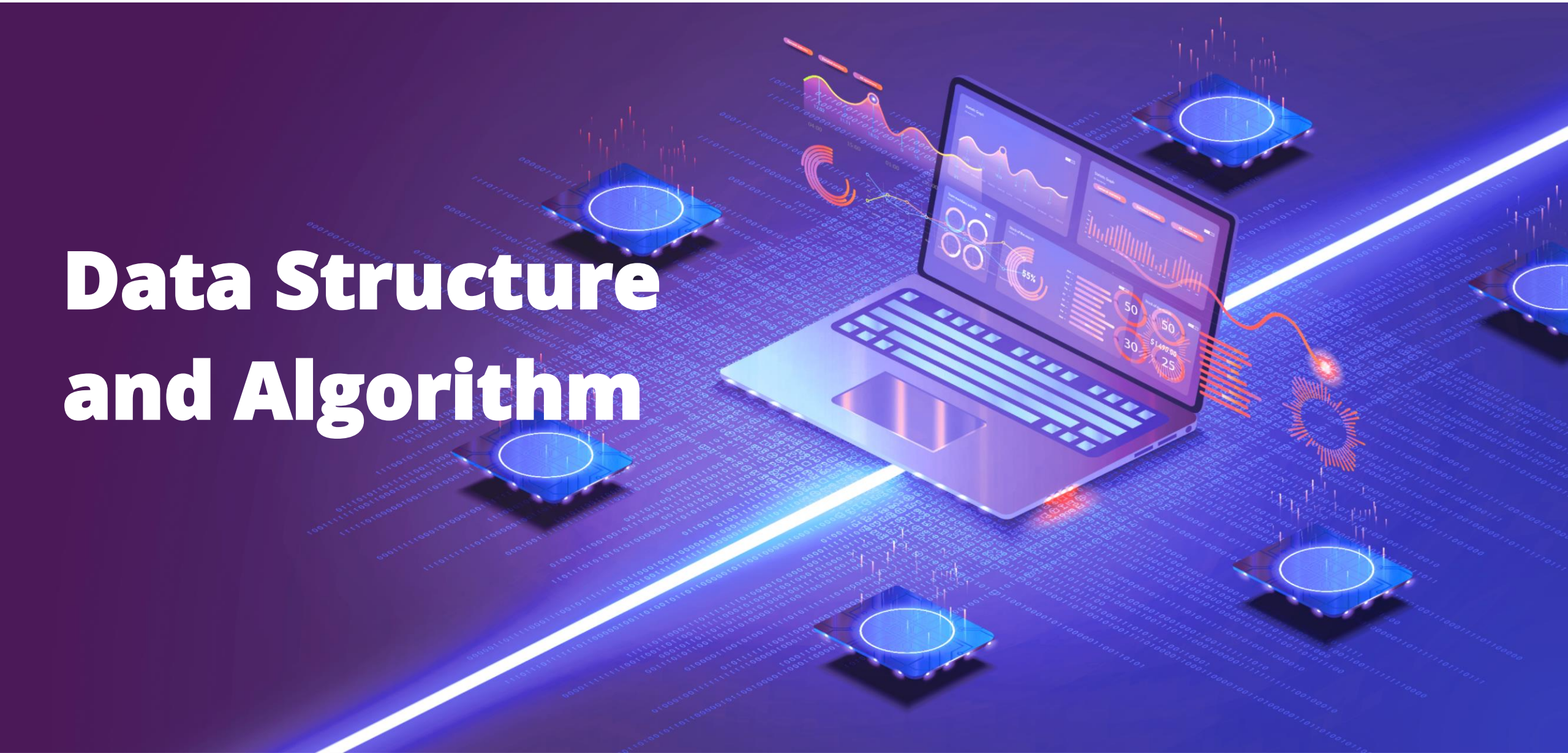
WILEY

ENABLING DISCOVERY | POWERING EDUCATION | SHAPING WORKFORCES

Data Structure and Algorithm



DATA STRUCTURES ALGORITHMS

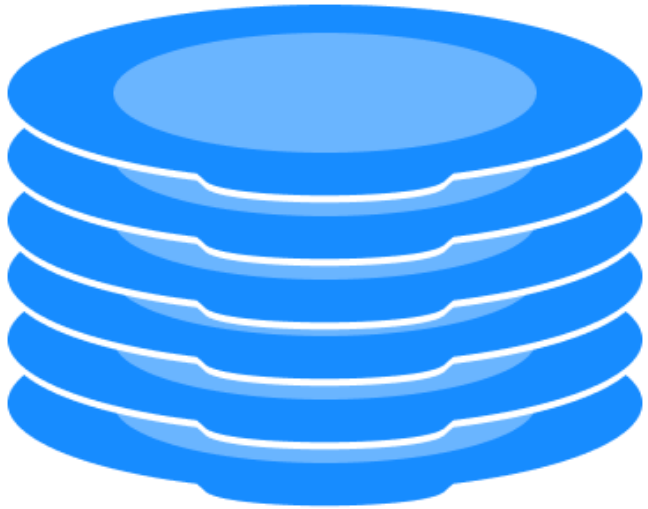
STACK

Stack Data Structure

In this tutorial, you will learn about the stack data structure and its implementation in Python, Java and C/C++.

A stack is a linear data structure that follows the principle of **Last In First Out (LIFO)**. This means the last element inserted inside the stack is removed first.

You can think of the stack data structure as the pile of plates on top of another.



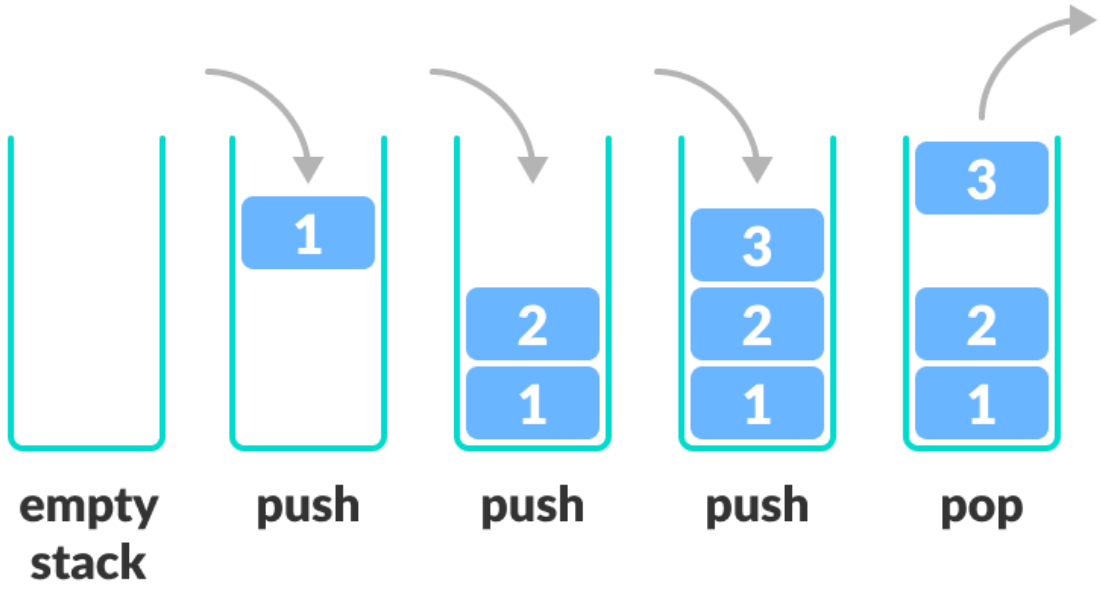
Here, you can:

- Put a new plate on top
- Remove the top plate

And, if you want the plate at the bottom, you must first remove all the plates on top. This is exactly how the stack data structure works.

Last in First Out Principle

In programming terms, putting an item on top of the stack is called **push** and removing an item is called **pop**.



In the above image, although item 3 was kept last, it was removed first. This is exactly how the LIFO (Last In First Out) Principle works.

We can implement a stack in any programming language like C, C++, Java, Python or C#, but the specification is pretty much the same.

Basic Operations on Stack

Push:

- Add an element to the top of a stack

Pop:

- Remove an element from the top of a stack

IsEmpty:

- Check if the stack is empty

IsFull:

- Check if the stack is full

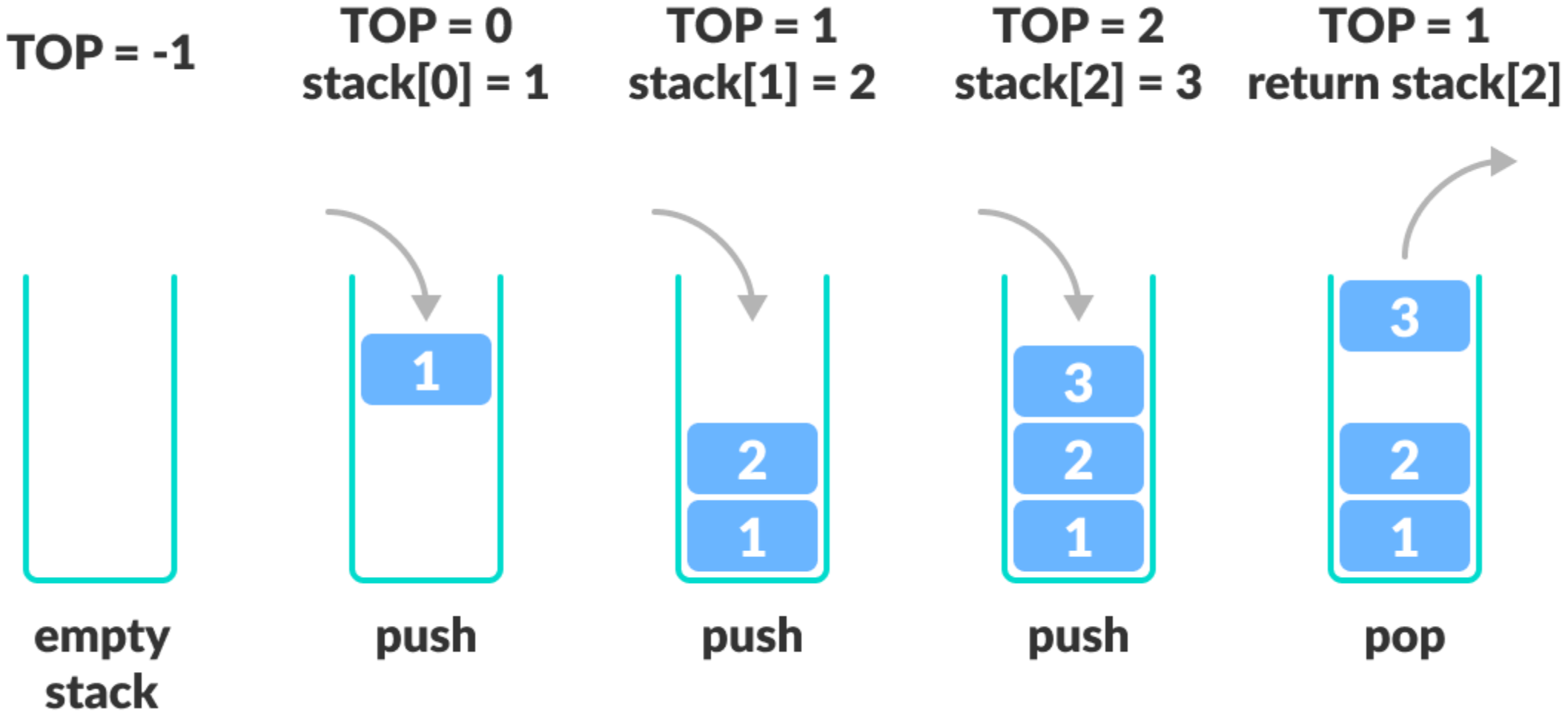
Peek:

- Get the value of the top element without removing it

Working of Stack Data Structure

The operations work as follows:

1. A pointer called `TOP` is used to keep track of the top element in the stack.
2. When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing `TOP == -1`.
3. On pushing an element, we increase the value of `TOP` and place the new element in the position pointed to by `TOP`.
4. On popping an element, we return the element pointed to by `TOP` and reduce its value.
5. Before pushing, we check if the stack is already full
6. Before popping, we check if the stack is already empty


TOP = -1
TOP = 0
stack[0] = 1
TOP = 1
stack[1] = 2
TOP = 2
stack[2] = 3
TOP = 1
return stack[2]
1
2
1
3
2
1
3
2
1
empty
stack
push
push
push
pop

Stack Implementation

Stack Time Complexity

For the array-based implementation of a stack, the push and pop operations take constant time, i.e. $O(1)$.

Applications of Stack Data Structure

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

- **To reverse a word** - Put all the letters in a stack and pop them out. Because of the LIFO order of stack, you will get the letters in reverse order.
- **In compilers** - Compilers use the stack to calculate the value of expressions like `2 + 4 / 5 * (7 - 9)` by converting the expression to prefix or postfix form.
- **In browsers** - The back button in a browser saves all the URLs you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack, and the previous URL is accessed.