# Real world problems

1. **File System Traversal**

Consider the task of traversing a file system, where you want to apply an operation (like search, delete, or modify) to each file

A screenshot of a computer

AI-generated content may be incorrect.

* *FileSystem:* collection contains *FileNode*objects.
* *FileNode:* a file/directory in the *FileSystem*.
* *Iterator<T>*: the iterator interface, implemented by *FileNodeIterator*.
* *FileNodeIterator:* iterator for *FileNode*objects in a *FileSystem*. It maintains its current position in the file system and implements *next* and *hasNext* methods.

🡪 The filesystem could be a tree structure or other complex format.

🡪 Using an iterator can *hide this complexity* and provide a *simple interface* *to go through each file or directory, one at a time.*

🡪 This makes the client code simpler and decouples it from the specific structure of the filesystem

[*https://cloudaffle.com/series/behavioral-design-patterns/iterator-pattern-application/#file-system-traversal*](https://cloudaffle.com/series/behavioral-design-patterns/iterator-pattern-application/#file-system-traversal)

1. **Social Media Feeds**

A diagram of a computer

AI-generated content may be incorrect.

* A *User* can have multiple *Post* instances.
* The *Feed* class aggregates posts from multiple users.
* The *PostIterator* implements the Iterator interface, providing a *unified way* to iterate over posts in a feed.
* This way, the client code (the code using these classes) does not need to be concerned with the internal structure of the *Feed* or *User* classes.

🡪 Social media feeds often involve large amounts of data.

🡪 Loading all this data at once may not be feasible or efficient.

🡪 Iterator pattern allows to load and process data in manageable chunks, providing a smooth user experience.

[*https://cloudaffle.com/series/behavioral-design-patterns/iterator-pattern-application/ - social-media-feeds*](https://cloudaffle.com/series/behavioral-design-patterns/iterator-pattern-application/#social-media-feeds)

# Relative with other patterns

1. **Composite + Iterator**

* Composite is used to build **tree structure** like a file system — folders can contain files or other folders

🡪 Instead of writing complex code to manually walk through each level of the tree, the **Iterator handles the traversal** for you.

1. **Factory Method + Iterator**

* You want different ways to iterate a collection like forward, reverse, or even skipping items

🡪 You can use Factory Method to **create specific iterator objects** without changing client code

A black background with white text

AI-generated content may be incorrect.

1. **Visitor + Iterator**

* Visitor separates the algorithm from the object structure. You can add new operations **without changing the elements themselves**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Without Visitor: you** have to **add a new method** to every element class each time you want a new operation
* **With Visitor:** You **don’t change the classes.** You add **new Visitor classes** to handle new operations
* You want to **apply operations** (like validation, export, logging) to every element in a structure

🡪 Iterator goes **through each element**, and **Visitor performs the action** on them

# Quizzes

1. **What is the main purpose of the Iterator design pattern?**
2. To add new behaviors to objects dynamically
3. To traverse elements of a collection without exposing its structure
4. To create families of related objects
5. To encapsulate operations as objects
6. **Which of the following is NOT a benefit of using the Iterator pattern?**
7. Hides the internal structure of the collection
8. Allows multiple traversals
9. Improves collection performance
10. Simplifies client code for traversal
11. **In which situation would an Iterator pattern be least appropriate?**
12. You want to traverse elements of a tree
13. You need to apply multiple traversal strategies
14. You’re using a single simple array for one-time iteration
15. You want to avoid exposing collection internals
16. **How does Iterator relate to the Composite pattern?**
17. Iterator replaces the Composite pattern
18. Iterator adds dynamic behavior to Composite
19. Iterator allows traversal of Composite structures uniformly
20. Iterator provides undo functionality for Composite objects
21. **What’s the correct way to introduce a new traversal strategy (e.g., reverse) in the Iterator pattern?**
22. Modify the original collection
23. Change the client logic
24. Create a new iterator class implementing the same interface
25. Rewrite the data storage logic