Iterators vs. Iterables

Question:

Iterators vs. Iterables

Recall from the video that:
- An iterable is an object that can return an iterator.
- An iterator is an object that keeps state and produces the next value when you call `next()` on it.

In this exercise, you need to identify which object is an iterable and which is an iterator by using `print()` and `next()`.

Possible Answers:
1. Both `flash1` and `flash2` are iterators.
2. Both `flash1` and `flash2` are iterables.
3. `flash1` is an iterable, and `flash2` is an iterator.

Correct Answer:

`flash1` is an iterable, and `flash2` is an iterator.

Explanation:

1. Iterable (`flash1`):
- `flash1` can be iterated over but does not maintain an internal state.
- Example: Lists, strings, and dictionaries are iterables because they can generate an iterator object using `iter()`.

2. Iterator (`flash2`):
- `flash2` is the object that keeps its state and returns the next value upon calling `next()`.
- Example: Once the iterator is created using `iter(iterable)`, calling `next()` on it will produce successive elements.

Code Demonstration:

```python
# Check `flash1`
print(flash1)          # Prints the iterable
print(next(iter(flash1)))  # Calls `iter()` to create an iterator, and `next()` fetches the first element

# Check `flash2`
print(flash2)          # Prints the iterator
print(next(flash2))    # Directly calls `next()` as `flash2` is already an iterator
```

Expected Output:

- For `flash1`:
  - Prints the contents of the iterable.
  - Calling `next(iter(flash1))` works because `flash1` is an iterable, and `iter()` converts it into an iterator.

- For `flash2`:
  - Directly prints the next item in the sequence as it is already an iterator.