Playwright Timeout Error - Root Cause and Fix

Error Summary:

```
playwright._impl._errors.TimeoutError: Locator.text_content: Timeout 30000ms
exceeded.
waiting for locator("//div[contains(text(),'\u20b9')]").nth(5)
```

Reason: This error occurs when Playwright is instructed to access an element at a specific index (e.g., `prices.nth(5)`) but that element **does not exist or is not ready** within the default timeout (30 seconds).

In scraping situations like Flipkart, the number of titles and prices may not always match due to:

- Lazy loading or asynchronous rendering
- Ads or non-product elements
- Missing prices for some items

Additionally, if you use `for i in range(await titles.count())` and assume the same number of prices exists, but Flipkart shows **fewer prices than titles**, then accessing `prices.nth(i)` will result in a **TimeoutError** if index `i` exceeds the available price elements.

Code Example with Problem:

```
count = await titles.count()
for i in range(count):
    title = await titles.nth(i).text_content()
    price = await prices.nth(i).text_content()  # <-- may raise timeout if i >=
prices.count()
```

Fix: Use Python's `min()` function to ensure you only iterate over the number of items that are **safely available** in both locators:

```
title_count = await titles.count()
price_count = await prices.count()
loop_count = min(title_count, price_count)  # Safest common range

for i in range(loop_count):
    title = await titles.nth(i).text_content()
    price = await prices.nth(i).text_content()
    # process data
```

What `` does: Returns the smallest value among its inputs.

Example:

```
min(10, 7)  # returns 7
```

This prevents index out-of-range errors or timeouts when one list is shorter.

Best Practice:

- Always compare element counts when working with multiple locators that should align by index.
- Optionally, use try-except to catch and log errors without breaking the whole loop.

Optional Robust Version:

```
for i in range(await titles.count()):
    title = await titles.nth(i).text_content()
    try:
        price = await prices.nth(i).text_content()
    except:
        print(f"Price not found for item {i}")
        continue

    if title and price:
        # store or process item
```

Next Button Loop Bug - Root Cause and Fix

Problem: On Flipkart's product listing pages, the "Next" button appears even on the last page. Clicking it again does **not** advance to a new page — the product listings remain the same, leading to an **infinite loop** in your scraper.

Reason: Flipkart's "Next" button:

- May be visible even when pagination has ended
- Does not change the URL
- Reloads the **same product data** (or does nothing) after the last page

Fix: Detect whether the page content changed after clicking "Next" by comparing the first product title **before and after** the click:

```
# Save the first product title
first_title_before = await titles.first.text_content()
```

```
# Click the Next button
await next_locator.click()
await page.wait_for_timeout(3000)

# Get first title after clicking
titles = page.locator("//div[contains(text(),'Laptop')]")
first_title_after = await titles.first.text_content()

# Check if you're stuck
if first_title_before == first_title_after:
    print("No change after clicking next - last page reached")
    break
```

Benefit: Prevents your scraper from reloading the same page in a loop when there's no new data to scrape.

Best Practice:

- Always verify if pagination actually changes content (not just the appearance of a button)
- Consider adding a `page_number` counter and logging progress during scraping