Selector Reference Guide for Playwright Scraping

🧠Understanding Selectors in Playwright

Selectors help identify HTML elements on a page for scraping. Choosing the right selector ensures your code works reliably even if the page layout changes slightly.

---

1. Basic CSS Selectors

- `tag` — Selects all elements by tag:

```
page.locator("div")
```

- `.class` — Selects by class name:

```
page.locator(".product-title")
```

- `#id` — Selects by ID:

```
page.locator("#price-container")
```

- `[attr=value]` — Selects elements by attribute:

```
page.locator("a[title]")  # all <a> tags with title attribute
page.locator("input[type='text']")
```

---

2. Combining Selectors

- `div a` — an `<a>` inside `<div>`:

```
page.locator("div a")
```

- `.card .price` — a `.price` inside `.card`:

```
page.locator(".card .price")
```

---

3. Dynamic or Autogenerated Class Names

Avoid brittle class names like `.Nx9bqj123`. Instead:

- Use **text**:

```
page.locator("div:has-text('₹')")
```

- Or use regex + fallback text extraction:

```
divs = await product.locator("div").all_text_contents()
for text in divs:
    if "₹" in text:
        match = re.match(r"₹\d[\d,]*", text.strip())
```

---

4. XPath Selectors (Advanced)

- Starts with `//`
- Example to get price:

```
page.locator("//div[contains(text(),'₹')]")
```

---

5. Locating Children or First Match

```
product = products.nth(i)
title = await product.locator("a[title]").first.text_content()
```

---

6. Text Content Check (Conditionals)

```
rating_locator = product.locator("span[id^='productRating'] div")
if await rating_locator.count() > 0:
    rating = await rating_locator.first.text_content()
```

👉Best Practices

- Prefer attribute-based selectors: `a[title]`, `div[data-id]`

- Avoid auto-generated class names
- Use `.first` to avoid ambiguity
- Always check `.count()` before `.text_content()` to prevent errors
- Use fallback text matching for dynamic content

This selector guide helps you build stable, maintainable scraping logic in Playwright.