split, and the Dot That Consumes Everything

split takes a regular expression, not a plain separator, so splitting on an unescaped dot matches every character and leaves nothing behind. It fails quietly -- no error, just an array that's wrong.

Automation Foundations · ashishautomationlabs.store

Test data arrives in rows. `split` cuts a String into an array on a separator:

```
String row = "LoginTest,PASS,240";
String[] parts = row.split(",");

fields: 3
name  : LoginTest
status: PASS
ms    : 240
```

That is a CSV row parsed into fields, and combined with Chapter 7 you can turn each row into a `TestResult` object. It is also where a trap waits, and it catches nearly everyone once:

```
String path = "a.b.c";
System.out.println("dot split wrong: " + path.split(".").length);
System.out.println("dot split right: " + path.split("\\.").length);

dot split wrong: 0
dot split right: 3
```

Splitting on a dot returned zero fields. The reason is that `split` does not take a plain separator -- it takes a **regular expression**, and in that language a dot means "any character." So the string was split on every character, leaving nothing behind. Escaping it as `"\\."` says "an actual dot" and produces the three fields you expected.

The characters that need escaping are the ones with special meaning in regular expressions: `. | ( ) [ ] { } ^ $ * + ? \`. A pipe-separated file split on `"|"` fails the same way, and it fails quietly -- you get an empty or nonsensical array rather than an error.

One more edge worth knowing: by default `split` discards empty strings at the *end* of the result. A CSV row ending in two commas can therefore produce fewer fields than you expect. Pass a negative limit -- `row.split(",", -1)` -- to keep them, which matters when column positions must line up.

`split` cuts a row apart on a pattern. The same pattern language does more than cutting -- it can also validate a whole value's shape, or strip everything that doesn't belong.