Snapshot Testing

Snapshot variants and running tests in CI.

What is a snapshot test?

A test that compares output of a test against a reference file.

- ✅ If they're the same, test passes.
- ❌ If they're different, test fails and you need to:
  - Reject the change and fix the code,
  - Or accept the change and save the new output as the new reference.

With `testthat::expect_snapshot` we can easily get started with snapshot testing.

- On first test run it will produce a snapshot file.
- On subsequent test runs it will compare output against the snapshot.

Use snapshot tests to assert on objects that are difficult to describe with code, like images.

But image output may be brittle.

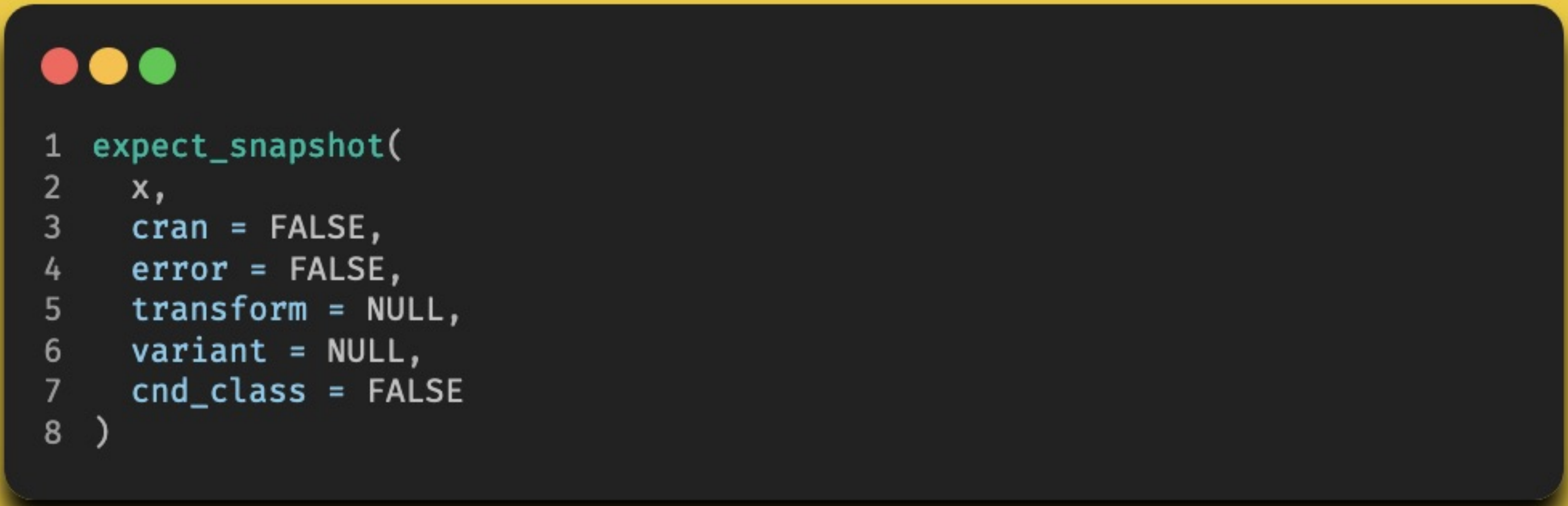
There may be slight rendering differences between machines making tests fail without any code changes.

Record information about dependencies used to produce snapshots.

We can produce variants of snapshots that depend on the OS and the R version.

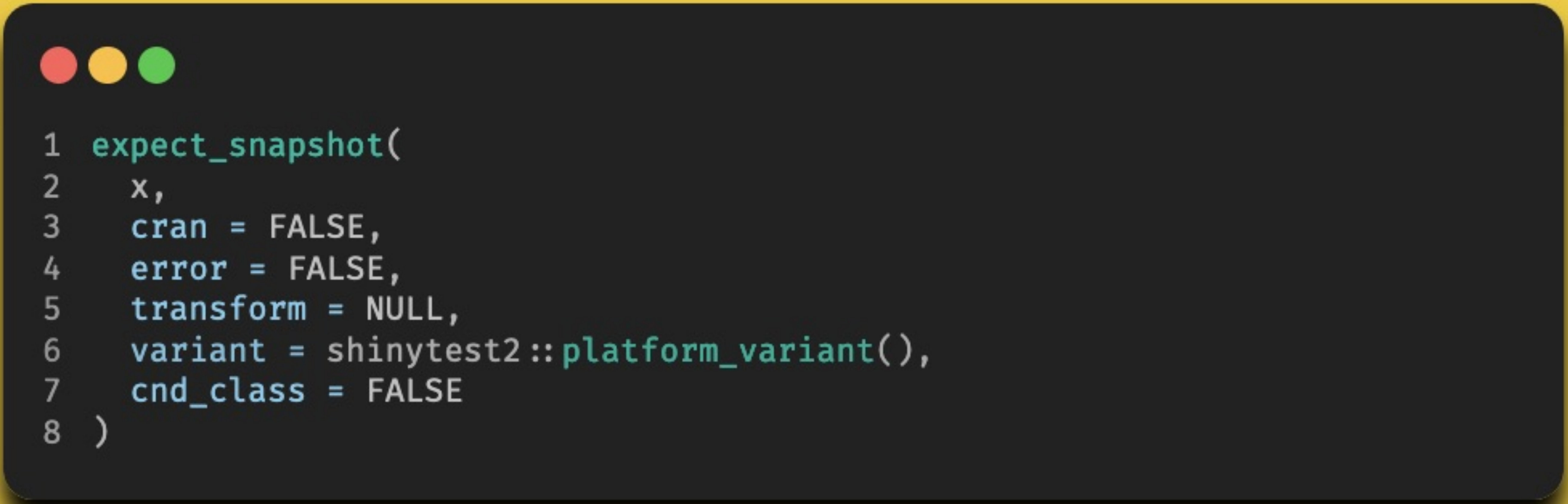
We could also include additional information like version of the data used in tests.

Use `variant` argument to record information about dependencies used to produce snapshots.



```r
expect_snapshot(
  x,
  cran = FALSE,
  error = FALSE,
  transform = NULL,
  variant = NULL,
  cnd_class = FALSE
)
```

Use `shinytest2::platform_variant` to capture OS and R version.



```r
expect_snapshot(
  x,
  cran = FALSE,
  error = FALSE,
  transform = NULL,
  variant = shinytest2::platform_variant(),
  cnd_class = FALSE
)
```

Variants are stored in separate directiories. Outputs are checked against their corresponding variants.

```
├── tests
│   ├── testthat
│   │   ├── _snaps
│   │   │   ├── variant_1
│   │   │   │   ├── test-file_1.md
│   │   │   │   ├── test-file_2.json
│   │   │   │   ├── test-file_3.png
│   │   │   ├── variant_2
│   │   │   │   ├── test-file_1.md
│   │   │   │   ├── test-file_2.json
│   │   │   │   ├── test-file_3.png
```

If two team members are using different systems, when they run tests they will produce 2 variants.

```
├── tests
│   ├── testthat
│   │   ├── _snaps
│   │   │   ├── mac-4.3
│   │   │   │   ├── test-file_1.md
│   │   │   │   ├── test-file_2.json
│   │   │   │   ├── test-file_3.png
│   │   │   ├── linux-4.3
│   │   │   │   ├── test-file_1.md
│   │   │   │   ├── test-file_2.json
│   │   │   │   ├── test-file_3.png
```

This approach ensures that snapshots are compared against files produced with similar dependencies.

It makes the system more robust.

How to use it with CI?

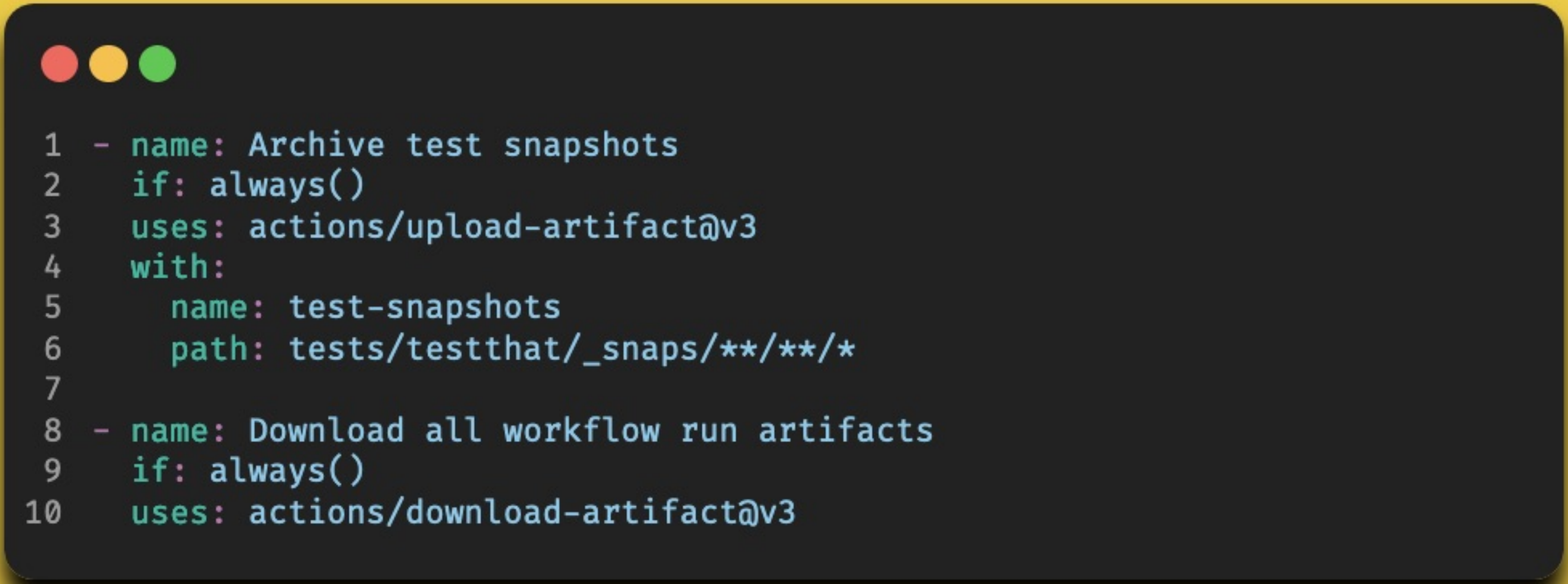
You may use macOS for development, but use linux to run the CI pipeline.

If we don’t use variants CI may fail because it will try to compare snapshots from different systems.

If we only have macOS variant, CI on each run will produce a snapshot without assertion.

We need to run the pipeline, save the snapshots and rerun the pipeline to compare the snapshots.

Add those steps to your CI pipeline, after a step that runs tests.



```yaml
- name: Archive test snapshots
  if: always()
  uses: actions/upload-artifact@v3
  with:
    name: test-snapshots
    path: tests/testthat/_snaps/**/**/*

- name: Download all workflow run artifacts
  if: always()
  uses: actions/download-artifact@v3
```

They will archive the snapshots and allow you to download them from `Actions` tab in Github.

```yaml
- name: Archive test snapshots
  if: always()
  uses: actions/upload-artifact@v3
  with:
    name: test-snapshots
    path: tests/testthat/_snaps/**/**/*

- name: Download all workflow run artifacts
  if: always()
  uses: actions/download-artifact@v3
```

The development cycle

1. Run the pipeline and produce new snapshots.
2. Go to the `Actions` tab and download the artifact.
3. Check-in new snapshots to the repository.
4. Push the changes.
5. Run the pipeline, now it asserts on proper snapshots.

Snapshot testing is easy and very powerful.

Use it to speed up your development and improve your confidence that your code works as expected.