# BLOCK NONCE #5: Don't make your tests too DRY

by guy.f@

## What?

The DRY principle ("Don't Repeat Yourself") encourages code reuse rather than duplication. But is it also the best thing for tests?

Read the test below and try to understand what it's doing and why it should pass:

```
#[test]
fn verify_balance_on_transfer() {
  let mut (account1, account2) = create_default_accounts();  // Test helper method.
  make_transfer(&mut account1, &mut acccount2);              // Test helper method.
  assert_eq!(account2.balance(), 1500);
}
```

## Why?

While the test body above is concise, the reader cannot understand what it does without looking at other places. Since tests don't have tests, it should be easy for humans to manually inspect them for correctness, even at the expense of some code duplication.

## How?

Make your tests **DAMP**! (Descriptive and Meaningful Phrases).
Read the new test below and try to understand what it's doing and why it should pass:

```
#[test]
fn verify_balance_on_transfer() {
  const INITIAL_BALANCE : u32 = 1000;
  const TRANSFER_AMOUNT : u32 = 500;

  let mut new_account =
      Account::new(AccountType::NO_OUTGOING_TRANSFERS(INITIAL_BALANCE));
  let mut account_to_transfer_from =
      Account::new(AccountType::SUPPORTS_TRANSFERS(TRANSFER_AMOUNT));

  assert_eq!(new_account.balance(), INITIAL_BALANCE);
  account_to_transfer_from.transfer_to(&mut new_account, TRANSFER_AMOUNT);
  assert_eq!(new_account.balance(), INITIAL_BALANCE + TRANSFER_AMOUNT);
}
```

**TIP**💡 The DRY principle is still relevant in tests. There is a trade-off between readable and unique. Try to avoid duplicate code (DRY) while keeping the code expressive and easy to verify (DAMP). DRY and DAMP sometimes conflict. In tests, usually lean towards using DAMP.

**Call For Action!**📢💨 Choose one DRY test from your repo and make it DAMP.

#starkcode