3 Ways to Name Parameters in Swift Parametrised Tests

MAKE YOUR TESTS SUPER-CLEAR

Maciej Gomółka

Option 1: First Named Tuples

Only the *first* tuple is named, all others rely on positional matching to
(a, b, result).

✅ Minimal boilerplate for small input sets
❌ Readability drops after adding more cases
❌ Easy to mix up arguments position
❌ Hard to scan or extend

```
@Test(arguments: [
    (a: 1, b: 2, result: 3),
    (10, 15, 25),
    (1, -5, -4),
    (-1, -5, -6),
    (0, 0, 0),
    (1000, 1000, 2000),
    (10000, 50000, 60000),
    (-10, -3, -13),
])
func add_returnsCorrectSum(a: Int, b: Int, result: Int) {
    #expect(add(a, b) == result)
}
```

Maciej Gomółka

Option 2: Named Tuples

Every tuple entry explicitly names *all* its fields *(a: ..., b: ..., result: ...)* for all cases.

✅ Clear - no guessing which value is which
✅ Easy to reorder or remove individual cases
❌ More repetitive boilerplate per line
❌ Becomes long and repetitive with many cases

```
@Test(arguments: [
    (a: 1, b: 2, result: 3),
    (a: 10, b: 15, result: 25),
    (a: 1, b: -5, result: -4),
    (a: -1, b: -5, result: -6),
    (a: 0, b: 0, result: 0),
    (a: 1000, b: 1000, result: 2000),
    (a: 10000, b: 50000, result: 60000),
    (a: -10, b: -3, result: -13),
])
func add_returnsCorrectSum(a: Int, b: Int, result: Int) {
    #expect(add(a, b) == result)
}
```

Maciej Gomółka

Option 3: Struct

Use a TestCase *struct* and pass it to @Test

✅ Perfect separation of data vs testing logic
✅ Great readability and maintainability
✅ Unlimited scalability
✅ IDE driven field suggestions
❌ More upfront work (struct definition)
❌ Might be overkill for 2-3 test cases

```
struct TestCase {
    let a: Int
    let b: Int
    let result: Int

    static var all: [TestCase] {
        [
            .init(a: 1, b: 2, result: 3),
            .init(a: 10, b: 15, result: 25),
            .init(a: 1, b: -5, result: -4),
            .init(a: -1, b: -5, result: -6),
            .init(a: 0, b: 0, result: 0),
            .init(a: 1000, b: 1000, result: 2000),
            .init(a: 10000, b: 50000, result: 60000),
            .init(a: -10, b: -3, result: -13)
        ]
    }
}

@Test(arguments: TestCase.all)
func add_returnsCorrectSum(testCase: TestCase) {
    #expect(add(testCase.a, testCase.b) == testCase.result)
}
```

Maciej Gomółka

Options Comparison

Criteria	**Option 1** First Named Tuple	**Option 2** Named Tuples	**Option 3** Struct
Readability	**Poor** (at scale)	**High**	**Highest**
Scalability	**Poor**	**Fair**	**Excellent**
Boilerplate	**Low**	**Medium**	**Medium** (upfront)
Test arguments number	**3**	**3**	**1**

Maciej Gomółka

My final advice

No One-Size-Fits-All

No single strategy works for every test suite. Evaluate pros and cons of each strategy and choose the one that best fit your needs.

Remember - Tests are not just checks, they're living documentation of your production code.
Keep them super clear and don't hesitate to ask your peers for feedback.

Maciej Gomółka