Minitest Cheat Sheet – The Basics

Assert-Style Testing

Test cases are written as standard Ruby classes inheriting from `Minitest::Test`. Public instance methods matching the pattern `test_*` are treated as tests.

```
class TpsReportTest < Minitest::Test
  def test_report_must_have_cover_sheet
    report = TpsReport.new(cover_sheet: nil)
    assert report.invalid?
  end
end
```

`setup` – Executes before each test

```
def setup
  @report = TpsReport.new(cover_sheet: true)
end
```

`teardown` – Executes after each test

```
def teardown
  @report = nil
end
```

`skip` – Ends the test immediately, result is neither pass or fail

```
def test_memo_received
  skip "Hasn't been written yet"
  # possible other assertions...
end
```

`flunk` – Fail the test immediately

```
def test_tps_report_has_cover_sheet
  flunk "No cover sheet ever"
end
```

Mocking and Stubbing

`Mock#expect` – Defines a method expectation on a mock object

```
mock_report = Minitest::Mock.new
mock_report.expect(:date, Date.today)
```

`Mock#verify` – Check that all method expectations were fulfilled

```
mock_report.verify
```

`Object#stub` – Replace a method within the scope of a block

```
report.stub(:date, Date.today) do
  assert_equal Date.today, report.date   # passes
end
```

Spec-Style Testing

Specs are written using a specialized DSL similar to the one used by RSpec. Test cases and tests are defined using block helper methods, and assertions use a syntax meant to more closely mirror natural language.

```
describe TpsReport do
  it “must have a cover sheet” do
    report = TpsReport.new(cover_sheet: nil)
    expect(report).must_be :invalid?
  end
end
```

`before` – Executes before each test in a given describe scope

```
before do
  @report = TpsReport.new(cover_sheet: true)
end
```

`after` – Executes before each test in a given describe scope

```
after do
  @report = nil
end
```

`describe` – Defines a scope for a collection of related tests

```
describe TpsReport do
  describe “Validations” do
    # …
  end
end
```

`it` – Defines a test method body

```
it “includes all TpsReports for the week” do
  @inbox.must_include @peters_report
end
```

`let` – Creates a named lazy initializer using the requested block

```
let(:report) { TpsReport.first }
```

`subject` – Creates a lazy initializer named subject

```
subject { TpsReport.first }
```

`expect` – Wrap a value before making expectations on it
Aliased as: _, **`value`**

```
expect(@tps_report.date).must_equal Date.today
```

Minitest Cheat Sheet – Assertions

Assertion / Refutation	Description	Examples
`assert/refute`	Returns a “truthy” / “falsy” value	`assert @report.has_cover_sheet?, “No cover sheet”`
`assert_empty/refute_empty`	Responds to `#empty?` and returns `true` / `false`	`assert_empty @inbox`
`assert_equal/refute_equal`	Expected and actual values are equal (`==`)	`assert_equal :paper_jam, @printer.current_status`
`assert_in_delta` `refute_in_delta`	Values' absolute difference falls in specified range	`assert_in_delta Math:: PI, (22.0 / 7.0), 0.01`
`assert_in_epsilon` `refute_in_epsilon`	Values' relative difference falls in specified range	`assert_in_epsilon 22.55, @item.price / 2.0, 0.01`
`assert_includes` `refute includes`	Collection includes the requested object	`assert_includes @calendar.days_of_week, "Monday"`
`assert_instance_of` `refute_instance_of`	Requested object is an instance of the given class	`assert_instance_of String, "Initech"`
`assert_kind_of` `refute_kind_of`	Requested object inherits from the class/module	`assert_kind_of Numeric, @report.page_count`
`assert_match/refute_match`	RegExp argument matches the actual String	`assert_match /synergies/, @report.text`
`assert_mock`	All mock expectations have been satisfied	`assert_mock @report_service_mock`
`assert_nil/refute_nil`	Tested object is `nil`	`assert_nil @report.cover_sheet`
`assert_operator` `refute_operator`	Binary expression prepared using the given arguments evaluates to `true`	`assert_operator @report.page_count, :>=, 20`

`assert_output`
`assert_silent`
	Block produces the expected output on `$stdout` or `$stderr` or remains silent on both	`# exact match on $stdout`
		`assert_output("OK") { system("echo OK") }`
		`# partial match with Regexp on #stderr`
		`assert_output("", /borked/i) { system("status") }`
		`assert_silent { system "find . -name *~ -delete" }`

`assert_predicate` `refute_predicate`	Message composed using the parameters returns true when sent to the tested object	`assert_predicate @report, :submitted?`
`assert_raises`	Block raises an error like the one specified	`assert_raises(ReportFormatError) { @report.submit }`
`assert_respond_to` `refute_respond_to`	Tested object `#responds_to?` requested message	`assert_respond_to "Bill", :length`
`assert_same/refute_same`	Both parameters refer to the same object	`assert_same @new_report, @report_from_db`
`assert_send`	Message sent to the receiver with the requested arguments returns `true`	`assert_send [ @calendar, :no_meetings?, :saturday]`
`assert_throws`	Block throws the expected symbol	`assert_throws(:found) { @haystack.find(“needle”) }`

Minitest Cheat Sheet – Expectations

Expectations	Description	Examples
`must_be/wont_be`	Statement prepared using the requested arguments evaluates to a "truthy" / "falsy" value	`expect(@report).must_be :complete?, "Incomplete"` `expect(@report.date).must_be :>, Date.today`
`must_be_empty` `wont_be_empty`	Responds to `#empty?` and returns `true` / `false`	`expect(@inbox).must_be_empty`
`must_equal` `wont_equal`	Expected and actual values are equal (`==`)	`expect(@printer.status).must_equal :paper_jam`
`must_be_within_delta` `wont_be_within_delta`	Values' absolute difference falls in specified range Alias: `must_be_close_to`/`wont_be_close_to`	`expect(22.0/7.0).must_be_within_delta Math::PI, 0.01`
`must_be_within_epsilon` `wont_be_within_epsilon`	Values' relative difference falls in specified range	`expect(@price/2.0).must_be_within_epsilon 22.55, 0.01`
`must_include` `wont_include`	Collection includes the requested object	`expect(@calendar.days_of_week).must_include "Monday"`
`must_be_instance_of` `wont_be_instance_of`	Requested object is an instance of the given class	`expect("Initech").must_be_instance_of String`
`must_be_kind_of` `wont_be_kind_of`	Requested object inherits from the class/module	`expect(@report.page_count).must_be_kind_of Numeric`
`must_match/wont_match`	RegExp argument matches the actual String	`expect(@report.text).must_match /synergies/`
`must_be_nil/wont_be_nil`	Tested object is `nil`	`expect(@report.cover_sheet).must_be_nil`

`must_output`	Output captured from `$stdout` / `$stderr` matches the expected output	`# exact match on $stdout`
		`callable = -> { system("echo OK") }`
		`expect(callable).must_output("OK")`
		`# partial match with Regexp on #stderr`
		`callable = -> { system("service-status") }`
		`expect(callable).must_output("", /borked/i)`

`must_raise`	Callable raises an error like the one specified?	`callable = proc { "TPS Report".submitted? }` `expect(callable).must_raise(NoMethodError)`
`must_respond_to` `wont_respond_to`	Tested object `#responds_to?` requested message	`expect("Bill").must_respond_to :length`
`must_be_same_as` `wont_be_same_as`	Both parameters refer to the same object	`expect(@report_from_db).must_be_same_as @new_report`
`must_be_silent`	Block produces no output on `$stdio` or `$stderr`	`callable = proc { system "find . -name *~ -delete" }` `expect(callable).must_be_silent`
`must_throw`	Block throws the expected symbol	`callable = -> { @haystack.search("needle") }` `expect(callable).must_throw(:found))`