Automated Testing
& Logging in Python – Advanced

Study Guide

1970-01-01

Pytest Advanced Cheat Sheet

- **Parametrize**: complex cases and ids.

```
@pytest.mark.parametrize("a,b,expected", [
    (6, 3, 2), (5, 2, 2.5)
], ids=["int", "float"])
def test_div(a,b,expected):
    assert a/b == expected
```

- **Markers**: `@pytest.mark.slow`, select via `-m slow`
- **Skip/XFail**: `@pytest.mark.skip`, `@pytest.mark.xfail`
- **Fixture scopes**: `function`, `class`, `module`, `session`

```
@pytest.fixture(scope="session")
def db_conn():
    yield connect()
    disconnect()
```

- **Autouse fixtures**: apply implicitly for setup like env vars.
- **Parallel runs**: `pytest -n auto` (requires `pytest-xdist`).

Pytest Execution Diagram

Logging Configuration (Production-Ready)

Use `dictConfig` for structured, multi-handler logging.

```
import logging, logging.config

LOGGING = {
  "version": 1,
  "disable_existing_loggers": False,
  "formatters": {
    "std": {"format": "%(asctime)s [%(levelname)s] %(name)s: %
```

```
(message)s"},
    "json": {"()": "pythonjsonlogger.jsonlogger.JsonFormatter"}
  },
  "handlers": {
    "console": {"class": "logging.StreamHandler", "formatter": "std"},
    "file": {
      "class": "logging.handlers.RotatingFileHandler",
      "filename": "app.log", "maxBytes": 1048576, "backupCount": 3,
      "formatter": "std"
    }
  },
  "root": {"level": "INFO", "handlers": ["console", "file"]}
}

logging.config.dictConfig(LOGGING)
logger = logging.getLogger(__name__)
logger.info("configured")
```

Tip: For JSON logs install `python-json-logger` and switch the formatter.

Pytest Logging Tips

- CLI live logs: `pytest -v —log-cli-level=INFO`
- Configure in `pytest.ini`:

  ```
  [pytest]
  log_cli = true
  log_cli_level = INFO
  ```

- Use `caplog.at_level(level, logger=name)` for scoped capture.

Selenium Patterns (Advanced)

Page Object Model (POM) skeleton:

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

class LoginPage:
    URL = "https://www.saucedemo.com/"
    USER = (By.ID, "user-name")
    PASS = (By.ID, "password")
    BTN  = (By.ID, "login-button")

    def __init__(self, driver):
        self.driver = driver
```

```
    def open(self):
        self.driver.get(self.URL)

    def login(self, u, p):
        wait = WebDriverWait(self.driver, 10)

wait.until(EC.visibility_of_element_located(self.USER)).send_keys(u)

wait.until(EC.visibility_of_element_located(self.PASS)).send_keys(p)
        wait.until(EC.element_to_be_clickable(self.BTN)).click()
```

Wait Strategies Cheat Sheet

- `visibility_of_element_located`
- `element_to_be_clickable`
- `presence_of_all_elements_located`
- `url_contains` / `title_contains`

Advanced Logging Diagram

Flaky Test Mitigation

- Replace `sleep` with explicit waits.
- Isolate state via `tmp_path`/`monkeypatch`.
- Use retries sparingly (e.g., `flaky` or custom wrapper).
- Run in CI with deterministic seeds and timeouts.