Python Testing Techniques Cheat Sheet

Learn more about testing Python apps in our in-depth tutorial at realpython.com/blog/python/python-cli-testing/

"Lo-Fi" Debugging With Print

When to use print debugging:

- Simple objects
- Shorter scripts
- Seemingly simple bugs
- Quick inspections

Dive deeper:

- pprint - prettify printed objects

Pros:

- Rapid testing
- Easy to use

Cons:

- Most cases you have to run the whole program, otherwise:
- You need to add extra code to manually control flow
- You can accidentally leave test code when done, especially in complex code

Using a Debugger

When to use a Python debugger:

- More complex projects
- Difficult to detect bugs
- You need to inspect more than one object
- You have a rough idea of where an error is occurring, but need to zero in on it

Dive deeper:

- Conditional breakpoints
- Evaluating expressions while debugging

Pros:

- Control over flow of program
- Bird's-eye view of application state
- No need to know exactly where the bug is occurring

Cons:

- Difficult to manually watch very large objects
- Long-running code will take very long to debug

Unit Testing with Pytest and Mocks

When to use Python unit testing frameworks:

- Large, complex projects
- OSS projects

Helpful tools:

- Pytest fixtures
- deepdiff for comparing complex objects
- Mocker

Pros:

- Automates running tests
- Can catch many types of bugs
- Simple setup and modification for teams

Cons:

- Tedious to write
- Has to be updated with most code changes
- Won't replicate true application running

Integration Testing

When to use integration testing in Python:

- Always :-)
- Generally after other test methods, if they're employed.

Helpful tools:

- tox environment and test automation management

Pros:

- See how your application runs in real-world conditions

Cons:

- Larger applications can be difficult to accurately track data flow through
- Have to have test environments that are very close to production environments