Robot Framework Testing and Debugging Cheat Sheet

A comprehensive cheat sheet covering essential techniques for testing and debugging Robot Framework projects, including logging, debugging tools, and best practices for robust test automation.

Logging and Reporting

Basic Logging

Robot Framework provides built-in keywords for logging messages at different levels.

`Log` - Logs a message at the INFO level.
`Log Many` - Logs multiple messages at the INFO level.
`Log Variables` - Logs all Robot Framework variables.

Example:

```
Log  'This is an info message'
Log Many  'Message 1'  'Message 2'
Log Variables
```

Logging Levels

`DEBUG`	For detailed debugging information.
`INFO`	General information about test execution (default).
`WARN`	Warnings about potential issues.
`ERROR`	Errors that occurred during test execution.
`TRACE`	More detailed info than DEBUG. Used for troubleshooting.

Setting Log Level

You can set the log level using command-line options.

`--loglevel DEBUG` - Sets the log level to DEBUG.

Alternatively, you can set the log level programmatically.

```
Set Log Level  DEBUG
```

Debugging Techniques

Using `Pause Execution`

The `Pause Execution` keyword allows you to temporarily halt test execution and inspect the current state.

```
Pause Execution
```

This keyword will print a prompt in the console, allowing you to execute Robot Framework keywords and Python code interactively.

Interactive Debugging with `rpdb`

`rpdb` is a remote debugger for Python. You can integrate it into your Robot Framework tests to set breakpoints and step through the code.

1. Install `rpdb`: `pip install rpdb`
2. Import `rpdb` in your Python library.
3. Set a breakpoint using `rpdb.set_trace()`.

Example:

```
import rpdb

def my_keyword():
    rpdb.set_trace()
    # Your code here
```

Then, run your test and connect to the debugger using a client like `telnet localhost 4444`.

Using `Evaluate` for Debugging

The `Evaluate` keyword allows you to execute Python code directly within your Robot Framework tests. This can be useful for inspecting variables or performing calculations during debugging.

```
Evaluate  print(variable_name)
```

Handling Exceptions

Try-Except Blocks

Robot Framework allows you to handle exceptions using `Try`, `Except`, and `Finally` blocks.

```
Try
    Some Keyword That Might Fail
Except  Exception as e
    Log  'An exception occurred: ${e}'
Finally
    Log  'This will always be executed'
End
```

Handling Specific Exceptions

You can catch specific exceptions using the fully qualified name of the exception class.	`Except ValueError as e`
	`    Log  'A ValueError occurred: ${e}'`

You can use multiple `Except` blocks to handle different types of exceptions.	`Try`
	`    # Some code here`
	`Except ValueError`
	`    # Handle ValueError`
	`Except TypeError`
	`    # Handle TypeError`
	`End`

Using `Run Keyword And Expect Error`

The `Run Keyword And Expect Error` keyword allows you to verify that a specific keyword raises an expected error.

```
Run Keyword And Expect Error
ValueError:Invalid value  Some Keyword That
Should Fail
```

The expected error can be a specific error message, a regular expression, or an exception class name.

Best Practices for Debugging

Isolate Issues

When encountering a test failure, try to isolate the issue by running the failing test case individually. This can help narrow down the source of the problem.

Use the `-t` or `--test` option to run a specific test case.

```
robot --test 'Test Case Name' tests.robot
```

Use Descriptive Logging

Add descriptive log messages to your tests to provide context and information about what the test is doing. This can make it easier to understand the test flow and identify the cause of failures.

Include relevant variable values and expected outcomes in your log messages.

Review Test Reports

Robot Framework generates detailed HTML reports and logs after each test execution. Review these reports to identify failures, errors, and warnings.

The reports provide information about the execution time, status, and log messages for each test case.