Missing Semester DBD002

Debuggers and Testing

Anirudh Gupta

Introduction

What is a Debugging?

When your code does not behave as you would expect, means it has some errors AKA BUGS. Debugging is De-bug-ing.

So what are Debuggers?

Debuggers are tools that help you find and fix bugs in your code.

Why Debuggers?

Debuggers are not just for finding bugs, they are also useful for understanding how a piece of code works.

It's like PythonTutor but more complicated and more powerful!

Languages and Debuggers

Different languages have different debuggers. For example-

- gdb for C/C++
- pdb for Python

Let's Debug!

Methods of Debugging

To Debug, we have various methods, some of them are:

- print statements
- assert statements
- logging
- Using Debuggers like gdb, pdb, etc.

Error Handling is REALLY important!

Print statement Debugging

- You insert print statements in your code to see which line messes up.

Assert statement Debugging

- You add assert statements to check if a condition is true. This is REALLY POWERFUL since you can make sure what is allowed to go ahead in the code!

```python
def divide(a, b):
    assert b != 0, "Cannot divide by zero"
    return a/b
```

Logging

- Logging is a way to track events that happen when some software runs.

Let's look at the example present in the logging_example.py file, in repostory: https://github.com/databasedIISc/Missing-Semester

```
python3 logging_example.py
# Raw output as with just prints

python3 logging_example.py log
# Log formatted output

python3 logging_example.py log ERROR
# Print only ERROR levels and above

python3 logging_example.py color
# Color formatted output
```

> Note: Leaning how to use colors makes visuals better! :)

How Linux and other OS's use logging?

- Every software, website, databases, etc. use logging to track events.
- and in future, you will too!

Linux

- Most Linux applications uses /var/log directory to store logs.
- Linux has system log which stores logs of all system events.
- systemd AKA System Daemon is a system and service manager for Linux like which services are enabled and running.
- systemd places the logs under /var/log/journal in a specialized format and you can use the journalctl command to display the messages.
- For MacOS, you have /var/log/system.log as the system log location.
- On most UNIX systems you can also use the dmesg command to access the kernel log.

logger

- For logging under the system logs you can use the logger shell program.

```
logger "I like cats! Meow"
# For MacOS
log show --last 1m | grep Meow
# For Linux
journalctl --since "1m ago" | grep Meow
```

There exists more tools which are easier to use and more powerful than logger.

Debuggers

Let's use Debuggers!

We will see how to use debuggers, mainly use Python debugger.

Some Concepts

Debuggers are programs that let you interact with the execution of a program, allowing the following:

- Halt execution of the program when it reaches a certain line.
- Step through the program one instruction at a time.
- Inspect values of variables after the program crashed.
- Conditionally halt the execution when a given condition is met.
- And many more advanced features.

PDB

It has the following common commands and let's look into that!

- l(ist) - Displays 11 lines around the current line or continue the previous listing.
- s(tep) - Execute the current line, stop at the first possible occasion.
- n(ext) - Continue execution until the next line in the current function is reached or it returns.
- b(reak) - Set a breakpoint (depending on the argument provided).
- p(rint) - Evaluate the expression in the current context and print its value. There’s also pp to display using pprint instead.
- r(eturn) - Continue execution until the current function returns.
- q(uit) - Quit the debugger.

VS Code Debugging

As you saw Command line visualisation of PDB, we will use VS code, since it provides an easier UI for us.

```
Why make Debugging harder when it's already hard!
```

See the `debugging_example.py` file in the repository for the code.

What's the issue?

```
n - i - 2 -> n - i - 1
```

Linters and Statics Analysis

Linters? Never heard?

Linters are tools that analyze your code to find potential errors, style issues, and other problems.

- But they do it within the code itself, not by running it.
- Advance linters point it out within the editor itself.
- Examples for python - `pylint`, `flake8`, `black`, etc.

Let's see an example -

- `static_analysis.py` file in the repository.

Clealy we see the issues in the code, but we need not notice, if we have linters!

Linters for other languages

- `clang-tidy` for C/C++
- `shellcheck` for shell scripts
- `eslint` for JavaScript
- `golint` for Go
- `rust-analyzer` for Rust

and more....

More into (only) Python Formatters, Linters and LSP!

> LSP - Language Server Protocol

Every language has its own formatters, linters and LSPs. These tools let you -

- Format your code automatically
- Find errors in your code(like linters)
- Provide autocompletion and other IDE features
- Let you rename imports, variables, etc. easily
- AND MUCH MORE.... For real! It's GODLY!

Python

- `black` for formatting
- `pylint` for linting
- `mypy` for static type checking
- `pep8` for style checking
- `rope` for refactoring
- `jedi` for autocompletion

and more ...

LSP in Text Editors

Most text editors support LSPs, and sometimes have them inbuilt, like VS Code or JetBrains, after simple installation of Extensions.

Testing

Now before we end up debugging, we should also know what the error is? Sometimes we don't even know where the code failed?

For algorithm, Moodle says - "WRONG!" and you are like - "Where?"

Writing Tests

> You write tests before you write your code,
> otherwise it will test your patience!

- Testing is a way to ensure that your code works as expected for your made test cases.
- It's like minimum requirement for your code to work. It may still not work always for all inputs after passing your made test cases.

Python Testing

- `unittest` - Python's built-in testing framework.
- `pytest` - A third-party testing framework that is easier to use and more powerful than `unittest`.

Let's do some "Pytesting" right now!

🟩 Pytest

```
pip install pytest
```

- Now check out the `test_example.py` file in the repository.

You run the tests by running `pytest` by-

```
pytest test_example.py -v
```

- `-v` is for verbose output.
- `-h` for more options.

▨ SUS

We noticed the test failed for `-10, -4` case. So we need to fix it.

▨ TO KNOW MORE INTO PYTEST, Check out their official documentation!

Let's Get more into Testing!

Profiling

- Profiling is a way to measure the performance of your code.
- It helps you identify bottlenecks in your code. The slowest part of your code.

Timing

- You can use the `time` module to measure the time taken by your code.
- better way it to use python's `cProfile` module or `line-profiler` module.

```
pip install line_profiler
```

- Check out the `profiling_example.py` file in the repository.

```
# For cProfile
python -m cProfile profiling_example.py

# For line-profiler
kernprof -l -v profiling_example.py
```

You can see the clear difference in the output!

Timings for general processes

- There exists a command line tool called `time`, in Unix which tells you the real, sys, user, etc time taken by ANY PROCESS to run in the shell.

```
time python3 profiling_example.py
```

This gives you at the bottom, the time taken by `time` command.

▒▒ Profiling

- Profiling is a way to measure the performance of your code.
- It helps you identify bottlenecks in your code. The slowest part of your code.

▒▒ Memory Profiling

- You can use the `memory_profiler` module to measure the memory usage of your code.

```
pip install memory_profiler
```

Now run on the same code -

```
python -m memory_profiler profiling_example.py
```

You can see the memory usage of your code!

Conclusion

There are more tools and techniques to debug and test your code, but these are the basics.

More Exploration

- Check out `perf` command in linux for Flame graphs.
- `valgrind` for memory profiling.
- `gdb` for C/C++ debugging. (We can have a session after you guys learn C/C++)

> Hope you liked the session!

ANY QUESTIONS?