Exercise L16: Debugging using `pdb` and `gdb`

1 Objective

Students will use:

- `pdb` (Python Debugger) to analyze and debug a faulty square root implementation.
- `gdb` (GNU Debugger) to debug a Fortran program.

Both exercises introduce essential debugging commands.

2 Exercise 1: Debugging Python with `pdb`

2.1 Modified Code with Intentional Error

The following Python code has a deliberate mistake. Students must use `pdb` to identify and correct it. The code is available in the class repository in the 'debugging' folder under python codes.

Listing 1: Faulty Python Code - exercise_pdb.py

```
def sqrt2(x, debug=False):
    from numpy import nan
    if x == 0:
        return 0
    elif x < 0:
        return nan
    s = 1.0
    kmax = 100
    tol = 1.0e-14

    for k in range(kmax):
        if debug:
            print(f"At iteration {k} the value of s={s
                :20.15f}")
```

```
14          s0 = s
15          s = 0.5 * s + (x / s)
16          delta_s = s - s0
17          if (abs(delta_s / x) < tol):
18              break
19
20      if debug:
21          print(f"Finally, the value of s={s:20.15f}")
22
23      return s
24
25  if __name__ == "__main__":
26      import numpy as np
27      number = 2.0
28      npsqrt=np.sqrt(number)
29      my=sqrt2(number)
30      print(f"my sqrt of {number} is {my} and numpy
           version if {npsqrt}")
```

2.2 Instructions

1. Introduce an Automatic `pdb` Breakpoint

Modify the script by adding:

```
1           import pdb
2           pdb.set_trace()
```

Place these lines **inside the loop** before Line 15 where `s` is updated.

2. Run the Script in shell

Execute the script using:

```
python exercise_pdb.py
```

3. Use the Following `pdb` Commands

- `list`: View surrounding code.
- `p <var>`: Print variable values (e.g., `p s`, `p delta_s`).
- `up`, `down`: Move between stack frames.
- `step (s)`: Step into the next line of execution.

- `continue (c)`: Continue execution until the next breakpoint. Keep using it to see the convergence behaviour.

3 Exercise 2: Debugging Fortran with gdb

3.1 Faulty Fortran Code

The following Fortran program has an error. Students will use `gdb` to analyze and fix it. The code is available in a folder named 'debugging' with the Fortran codes.

Listing 2: Faulty Fortran Code - exercise_gdb.f90

```
program debug_example
    implicit none
    real :: a, b, c
    integer :: i

    a = 10.0
    b = 0.0
    c = a / b

    print *, "Result:", c
end program debug_example
```

3.2 Instructions

1. Compile the Fortran Code with Debug Symbols

Compile the program with the `-g` flag:

```
gfortran -g example_gdb.f90
```

2. Run the Debugger

Start `gdb` and load the executable:

```
gdb a.out
```

3. Set a Breakpoint and Run

Set a breakpoint at the division line:

```
break 7
run
```

4. Use `gdb` Commands to Debug

- `list`: Display the source code.
- `print <var>`: Check variable values (`print a`, `print b`).
- `backtrace`: Show the call stack.
- `continue`: Resume execution until the next breakpoint.
- `help`: Explore other commands availabel in gdb.