27. Debugging and Profiling

These libraries help you with Python development: the debugger enables you to step through code, analyze stack frames and set breakpoints etc., and the profilers run code and give you a detailed breakdown of execution times, allowing you to identify bottlenecks in your programs.

- 27.1. `bdb` — Debugger framework
- 27.2. `faulthandler` — Dump the Python traceback
  - 27.2.1. Dumping the traceback
  - 27.2.2. Fault handler state
  - 27.2.3. Dumping the tracebacks after a timeout
  - 27.2.4. Dumping the traceback on a user signal
  - 27.2.5. Issue with file descriptors
  - 27.2.6. Example
- 27.3. `pdb` — The Python Debugger
  - 27.3.1. Debugger Commands
- 27.4. The Python Profilers
  - 27.4.1. Introduction to the profilers
  - 27.4.2. Instant User’s Manual
  - 27.4.3. `profile` and `cProfile` Module Reference
  - 27.4.4. The `Stats` Class
  - 27.4.5. What Is Deterministic Profiling?
  - 27.4.6. Limitations
  - 27.4.7. Calibration
  - 27.4.8. Using a custom timer
- 27.5. `timeit` — Measure execution time of small code snippets
  - 27.5.1. Basic Examples
  - 27.5.2. Python Interface
  - 27.5.3. Command-Line Interface
  - 27.5.4. Examples
- 27.6. `trace` — Trace or track Python statement execution
  - 27.6.1. Command-Line Usage
    - 27.6.1.1. Main options
    - 27.6.1.2. Modifiers
    - 27.6.1.3. Filters
  - 27.6.2. Programmatic Interface
- 27.7. `tracemalloc` — Trace memory allocations
  - 27.7.1. Examples
    - 27.7.1.1. Display the top 10
    - 27.7.1.2. Compute differences
    - 27.7.1.3. Get the traceback of a memory block
    - 27.7.1.4. Pretty top
  - 27.7.2. API

- 27.7.2.1. Functions
- 27.7.2.2. DomainFilter
- 27.7.2.3. Filter
- 27.7.2.4. Frame
- 27.7.2.5. Snapshot
- 27.7.2.6. Statistic
- 27.7.2.7. StatisticDiff
- 27.7.2.8. Trace
- 27.7.2.9. Traceback