Python Debugging Cheat Sheet

7 Techniques That Find Bugs in Minutes

I timed myself debugging the same bug using 3 methods: **print() marathon**: 47 min | **pdb.set_trace()**: 12 min | **Technique #4 (icecream)**: 3 min. Here's what actually works.

1. breakpoint() — Built-in Since Python 3.7

Stop using `import pdb; pdb.set_trace()`. One-liner since Python 3.7:

```
def process_data(items):
    for item in items:
        breakpoint()  # drops into debugger
        processed = transform(item)

# In the debugger:
# n → next line  |  s → step into  |  c → continue
# p var → print  |  l → show context
# PYTHONBREAKPOINT=0  →  skip all breakpoints in CI/CD
```

2. logging Over print — Always

Toggle with `LOG_LEVEL=DEBUG python script.py` — zero code changes:

```
import logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s [%(levelname)s] %(filename)s:%(lineno)d — %(message)s'
)
log = logging.getLogger(__name__)

log.debug(f'Processing: {item!r}')      # dev only
log.info(f'Pipeline started: {len(x)} items')
log.warning('Rate limit at 80%')
log.error('API failed', exc_info=True)  # auto-captures stack trace
```

3. traceback.format_exc() — Catch Without Losing Context

```
import traceback, logging
log = logging.getLogger(__name__)

try:
    risky_operation()
except Exception as e:
    log.error(f'Failed: {e}')            # summary for alerts
    log.debug(traceback.format_exc())     # full trace in debug
    # or: raise  — re-raise with context intact
```

4. icecream — The Print Statement Upgrade ■

`pip install icecream` — auto-prints variable names + file:line:

```
from icecream import ic

# Before: print(f'data = {data}')
ic(data)  # ic| data: {'key': 'value', 'count': 42}

# Works on expressions:
ic(len(items), items[0], process(items[0]))
# ic| len(items): 100, items[0]: 'first', process(..): 'FIRST'

ic.disable()  # one line to silence all ic() calls
```

5. @dataclass — Free __repr__ Forever

```
from dataclasses import dataclass

@dataclass
class Pipeline:
    name: str
    items_processed: int = 0
    errors: list = None

p = Pipeline('content-pipeline', 150, ['timeout on item 23'])
print(p)
# Pipeline(name='content-pipeline', items_processed=150,
#          errors=['timeout on item 23'])
```

6. hunter — Surgical Line-by-Line Tracing

`pip install hunter` — trace execution without touching the code:

```
import hunter

hunter.trace(module='my_module', action=hunter.CodePrinter)
result = the_suspicious_function()
hunter.stop()

# Real use: traced 1,200 lines, found timezone bug in 8 min
```

7. faulthandler — Catch Crashes That Print Nothing

```
import faulthandler
faulthandler.enable()  # top of script — catches C-level crashes

# Or from CLI:
# python -X faulthandler script.py

# Saved me: numpy segfault with zero Python output
# faulthandler pointed straight at line 97
```

Quick Reference

Situation	Tool
Interactive debugging	breakpoint()
Production logging	logging module
Catching exceptions	traceback.format_exc()
Print-debug replacement	icecream
Object inspection	@dataclass
Line-by-line trace	hunter
Silent C-level crash	faulthandler

Based on real debugging sessions across 3 production Python services.
More free cheat sheets: jacksonlee71.gumroad.com