miniVivisection — Examples Guide

Learn debugging in Rhino with 17 small scripts

These scripts teach the miniVivisection plug-in, and debugging itself, one step at a time. Each script has a bug in it. You find the bug with the feature that the script is about, usually in a way that wasn't possible before. No debugging experience is needed. Start with 01 and go in order: each lesson builds on the ones before.

1. Debugging in two minutes

A **debugger** runs your script and stops it at a line you choose. While it is stopped (*paused*), you can look at every variable and see what the script really computed, not what you think it computed.

Term	What it means
Breakpoint	A red dot left of a line number. Click there to set or remove one. The script stops when it reaches that line.
Debug (F5)	Runs the script in the debugger, so it stops at breakpoints. (F5 without a breakpoint is a plain Run; miniVivisection's **Debug** button always debugs -- lessons 15–17 start with it, without a breakpoint.)
Paused	The script waits. The highlighted line is the **next** one to run; it has **not** run yet.
Continue (F5)	Runs on to the next breakpoint, or to the end.
Step over (F10)	Runs one line, then pauses again.
Step into (F11)	If the line calls a function, goes into it and pauses at its first line.
Step out (Shift+F11)	Finishes the current function and pauses where it was called.
Stop (Shift+F5)	Ends the run.
Break now (Ctrl+Alt+B)	miniVivisection: pauses a script running in the debugger at its next line -- for a script that hangs or loops.
Variables tab	The editor's list of the paused script's variables (bottom panel).

The basic method: guess where things go wrong, set a breakpoint just before that place, look at the values, step forward, and narrow it down until you see the line that makes a wrong value.

2. What miniVivisection adds

The Script Editor's own debugger shows values as text, and Rhino freezes while the script is paused. With miniVivisection, in short:

- **Geometry in the viewports:** variables are drawn in Rhino while the script is paused.
- **Values in the code:** values appear in the tooltips and at the ends of the lines.
- **Edit and keep:** you can change geometry with Rhino's tools and write it back into the script, or bake it into the document.
- **Smarter breakpoints:** breakpoints can pause only when a condition holds.
- **Prompts on the paused script:** Python, PowerShell and Claude work on the script while it waits.
- **Rhino stays usable** the whole time.
- **Ctrl+Alt+B** pauses a running script, and **Grasshopper** script components can be debugged the same way (manual, sections 8 and 9).

Its tabs and pages all start with **V** (V°Watch, V°Python, ...). The **V°Debug** switch in the editor's top bar must be ON.

3. How to do a lesson

1. In Rhino, run `ScriptEditor`. In its left panel, open **Examples** > **miniVivisection** and double-click a lesson (or **File** > **Open** it from this folder).

2. Read the text at the top of the file: **The bug**, **Try this** and **Without miniVivisection** (and a **Needs:** line under the title when the lesson needs something first).
3. Set a breakpoint on the line marked `# <-- BREAKPOINT`, and press **F5** -- unless the lesson's *Try this* says otherwise: lessons 15–17 start with Debug and no breakpoint; lesson 04 has two breakpoints.
4. Follow the steps. When you have found the bug, stop (Shift+F5), fix the code, and run again to check.

Important: what you change while the script is paused (dragged points, values typed into V°Python) lasts for this run only. To keep a fix, change the code.

Your copies: the lessons open from your own copies in `%LOCALAPPDATA%\miniVivisection\lessons`; a lesson you changed and saved keeps your version, and when a new miniVivisection brings a new version of that lesson, it is put beside yours as *(name) (new)*, for example `gh3_stop_a_hang (new).gh`.

4. The lessons

#	File	You learn	Debugging idea
01	`01_watch_geometry.py`	the V°Watch tab: variables drawn in Rhino	**Look, don't guess:** a picture shows what numbers hide.
02	`02_values_in_the_code.py`	tooltips, values at line ends, "was", Tree; stepping with F10	**Step and watch values change** until one goes wrong.
03	`03_bake_before_a_crash.py`	Bake	**Pause before a crash** and keep what was already computed.
04	`04_edit_in_rhino.py`	Edit in Rhino, Auto edit	**Try a fix on live data,** then copy it into the code.
05	`05_conditional_breakpoints.py`	conditions, hit counts, log messages	**Filter the noise:** pause only at the case that fails.
06	`06_watch_expressions.py`	expressions in watches	**Ask the paused script questions** without changing the code.
07	`07_raw_data_interpret_as.py`	drawing plain lists of numbers	**Data is geometry too:** draw it to check it.
08	`08_python_terminal.py`	the V°Python tab	**Experiment while paused**, then continue with the fix.
09	`09_step_over_other_files.py`	stepping into, over and out of other files	**Follow the call** into the function that computes the wrong value.
10	`10_visualizers.py`	drawing your own classes (`rhinoviv`)	**Make your own objects visible.**
11	`11_rhino_usable_while_paused.py`	using Rhino while paused	**Check the model** the script works on, while it waits.
12	`12_powershell_terminal.py`	the V°PowerShell tab	**Check the files** the script reads.
13	`13_claude.py`	the V°Claude tab	**Ask for a second opinion** that sees the real values.
14	`14_threads_async.py`	threads: V°Threads, `thread_name` in conditions, watches in every thread, the freeze guard	**One thread at a time:** pause the thread that matters, look at all the others.
15	`15_deadlock.py`	hangs: "Where is everyone?", Break now	**When nothing happens, ask where everyone is.**
16	`16_long_running_watch.py`	a long-running script: Break now while it runs, fix it in the paused frame (V°Python), let it run on	**Don't stop a running script to look at it -- pause it.**
17	`17_runtime_error.py`	a runtime error: the debugger stops at it, the values and the call stack say why; Stop and record keeps it for later	**Look at the numbers where it failed,** not only at the message.

Lesson 09 uses three files: keep them in the same folder. Lessons 03, 11, 16 and 17 add objects to the open document; use an empty model. Lesson 13 needs the Claude Code CLI and the **V°AI** switch ON. Lessons 14 and 15 start with `# async:true`: the script runs on a thread of its own, so Rhino stays free while its threads run and pause.

Grasshopper lessons

Twelve definitions in `grasshopper\`, each with its lesson in a Panel on the canvas (The bug, Try this, Without miniVivisection). The lesson's text is Markdown, shown rendered: *right-click the Panel > V° Show as > Plain text* shows its source. Open them in Grasshopper (**File > Open**) from **`%LOCALAPPDATA%\miniVivisection\lessons\grasshopper`** -- your own copies, made when Rhino loads miniVivisection; the originals next to the plug-in stay as they are. Under each lesson's text, a V°Protect: its *Revert* button puts the lesson back as it was shipped, green while nothing is changed.

Lesson	File	What you learn	Good habit
GH 1	`gh1_which_branch_failed.gh`	failing runs named on the component; copying their paths; Debug failing run N	**Ask which data failed,** not just that something did.
GH 2	`gh2_the_one_wrong_item.gh`	Always debug, a breakpoint condition in a script component, the run badge; moving a slider while paused	**Stop only at the item that's wrong.**
GH 3	`gh3_stop_a_hang.gh`	Ctrl+Alt+B with the component's editor closed	**A hang isn't a crash:** stop it and see where it was.
GH 4	`gh4_v_edit.gh`	V°Edit: editing Grasshopper geometry in Rhino	**Fix the shape by hand** where numbers are slow.
GH 5	`gh5_runtime_error.gh`	a runtime error in a script component: the debugger stops at it (Always debug), the call stack shows which round of the loop	**One run can hide many steps:** find the one that failed.
GH 6	`gh6_threads_in_a_component.gh`	threads in a script component: results in finishing order; debugging after the join; V°Threads; why a breakpoint inside a worker doesn't pause in Grasshopper (the freeze guard) -- and does with `# async:true`, in each thread in turn	**Give every thread its own place** for its result.
GH 7	`gh7_parallel_run.gh`	`ghpythonlib.parallel.run`: in Rhino 8 it flattens only .NET collections, so a list per curve reaches line 16; the debugger stops at the error; a breakpoint inside the worker function pauses once, for the first item	**Check what a library hands back** before you trust its options.
GH 8	`gh8_long_running.gh`	a component that computes for 15 s: Ctrl+Alt+B pauses it (Always debug), V°Watch draws what it has so far, a fix in V°Python, and it runs on; `# async:true` keeps Rhino and Grasshopper usable while it computes	**Don't stop a running script to look at it -- pause it.**
GH 9	`gh9_script_from_a_panel.gh`	a script that comes from a Panel: edited in the Script Editor, saved back into the Panel, the component recomputes	**Edit code in a code editor,** wherever it's kept.
GH 10	`gh10_script_from_a_file.gh`	a script file named in a Panel (Input Is Path): Ctrl+Alt+B stops the hang, and the file itself opens at the line where it was	**A hang tells you where it is** -- if you stop it and look.
GH 11	`gh11_markdown_notes.gh`	notes in Markdown, rendered in a Panel (*Show as* Markdown); a double-click edits the source where you clicked; a script Panel edited in place to match the notes; links -- click them -- to its components, its canvas view and the Rhino example (which opens in Rhino)	**Keep the notes next to the code** -- and check one against the other.
GH 12	`gh12_markdown_parser.gh`	Markdown with Rhino's parser (Markdig): task boxes, nested lists and quotes, a footnote; a task without its space and a table without its header	**Mind the details of the syntax** -- a

Lesson	File	What you learn	Good habit
		rule, fixed in the source; switching the parser, turning Markdown off	space, a rule row.

All examples at a glance: `00_overview.3dm` (Rhino) and `grasshopper\gh0_overview.gh` (Grasshopper) -- click a link -- list every lesson with a link that opens it -- a Python lesson in the Script Editor, a Grasshopper lesson in Grasshopper, the Markdown text example in Rhino. Both are protected and say so in their text (*protected*, *not modified*): in Rhino by the `ProtectModel` command -- move the V° logo, the line says *modified*; click *Revert to the original* --, in Grasshopper by **V°Protect** -- move its *Try me* slider, V°Protect turns red; click *Revert*.

The Markdown text example, in `00_overview.3dm` right of the overview (next to the lessons; your copy in `%LOCALAPPDATA%\miniVivisection\lessons`) -- Markdown made into Rhino text by the `MarkdownText` command. Open it and double-click any of the text: its Markdown opens in a dialog, the caret at the part you double-clicked; change it and press OK -- the same objects are updated. Click a link under *Links* (the mouse over one says where it goes): the named view *Overview* (all of the file) comes back, *Perspective* becomes the active view, the V° logo (its outlines, named *Logo*) is zoomed to and marked in red, and the GH 11 links open that lesson in Grasshopper -- its *Ring* component marked, its canvas view restored; GH 11's own links lead back here. Edit a word with Rhino's own text editor, or move the whole text, then double-click it again: the change is in the Markdown.

GH 3, GH 8 and GH 10 are saved with their **Run** toggle off -- set it back to False before you save them, or the definition hangs (or computes for long) when it opens. GH 4 needs the **V°Edit** component (`miniVivisection.GH.gha`, loaded with the plug-in). GH 10 runs `gh10_spiral.py`, named without a folder: keep the two files in the same folder.

5. If something doesn't work

- **The script doesn't stop:** is the red dot there? Does the script reach that line at all? Move the breakpoint up.
- **No V tabs, no values in the code:** turn **V°Debug** ON in the editor's top bar. Check that miniVivisection is loaded: its name and version show in the top bar.
- **A watch says it can't be evaluated:** the name doesn't exist at the paused line yet. Step on, or check the spelling.
- **"nothing to draw":** the value isn't geometry. Try **Interpret as** (lesson 07) or a visualizer (lesson 10).
- **The script starts with `#! python2`:** most features are for Python 3 only.
- **A breakpoint in a worker thread doesn't pause:** the script runs on Rhino's own thread, which would freeze. Add `# async:true` after `#! python 3` (lesson 14); the Terminal says so too.
- **No miniVivisection folder in Examples:** V°Settings > "Lessons in the Examples tab", and V°Debug ON.
- **More help:** `miniVivisection_Manual.pdf` in this folder (all features in short), and the **V°Log** page in the editor's left panel: what happened, in plain words.