Python's built-in debugger (PDB)

It is a powerful tool for debugging Python code, allowing you to set breakpoints, step through code, and inspect variables during runtime.

Step 1: Importing PDB

The first step is to import the PDB library. You can use it in two main ways:

- 'Manually': By adding 'import pdb; pdb.set_trace()' in your code where you want to start debugging.

- 'Command Line': By running your script with 'python -m pdb <script.py>' to start debugging right from the beginning.

Step 2: Basic Commands in PDB

Once the PDB prompt appears, you can use the following commands to navigate through your code:

Command	Description
'n' (next)	Move to the next line within the same function.
's' (step)	Step into a function call to see its inner workings.
'c' (continue)	Continue running until the next breakpoint.
'q' (quit)	Exit the debugger and terminate the program.
'p' (print)	Print the value of a variable or expression.
'l' (list)	Display the current lines of code in the script around your current position.
'h' (help)	Display help for commands in PDB.

Step 3: Setting Breakpoints

There are a few ways to set breakpoints:

1. Inline': Insert 'pdb.set_trace()' at the point in your code where you want to pause execution.

```
Python
 import pdb
   def calculate_sum(a, b):
       pdb.set_trace()  # Execution will pause here
       result = a + b
       return result
 or
import pdb
   def calculate_sum(a, b):
       breakpoint()  # Execution will pause here
       result = a + b
       return result
```

2. Execution will stop at pdb.set_trace() and breakpoint(), asking users to provide input in the command line.

Step 4: Inspecting Variables

To inspect variables or expressions, use 'p' or 'pp' (pretty-print) followed by the variable or expression:

```
Python
p variable_name
pp complex_structure  # for dictionaries, lists, etc.
```

Step 5: Moving Through Code

As you debug, you may want to navigate around your code. Here's how:

- 'Step into Functions ('s')': Use this to jump inside a function to examine its execution.

- 'Next Line ('n')': Use this to execute the current line and move to the next one.

- 'Return ('r')': Finish the current function and return to the calling line.

Example code:

```
Python
import pdb

def calculate_area(length, width):
    area = length * width
    pdb.set_trace()  # Debugger starts here
    return area

def main():
    length = 5
    width = 10
    area = calculate_area(length, width)
    print(f"Area: {area}")

main()
```

With this code:

1. Run the script: 'python script.py', or if you are running it in the colab cell, then run the cell.

2. When PDB stops at 'set_trace()', try using commands like **'p length'** and **'p width'** to know the value of length and width, '**n**' to go to the next line, or '**s**' to step inside a function.

Summary of PDB Commands

```
Unset
- 'c': Continue execution until the next breakpoint.
- 'n': Move to the next line in the current function.
- 's': Step into a function call.
- 'p <variable>': Print a variable's value.
- 'q': Quit the debugger.
```

Using PDB effectively can help identify and solve bugs quickly, making it a valuable tool for any Python developer.