Debugging Python code is the process of identifying and fixing errors or unexpected behaviour. There are several ways to debug Python code, ranging from simple print statements to professional debugging tools. Here's a breakdown number of way for debugging:

1. Using `print()` Statements (Basic)

This is the simplest method.

```
def divide(a, b):
    print(f"a = {a}, b = {b}")  # Debug info
    return a / b

result = divide(10, 2)
print("Result:", result)
```

- Good for quick checks
- Not ideal for complex issues

2. Using the `pdb` Module (Python Debugger)

Python has a built-in debugger called `pdb`.

```
import pdb

def divide(a, b):
    pdb.set_trace()  # Pauses execution here
    return a / b

result = divide(10, 0)
```

When it pauses, you can use commands like:

- `n`: next line
- `c`: continue
- `q`: quit
- `p variable`: print a variable's value
- `l`: list source code

3. Using IDE Debuggers (Recommended for Larger Projects)

Most modern IDEs like **VS Code**, **PyCharm**, or **Thonny** have built-in graphical debuggers.

In VS Code:

1. Open your `.py` file.
2. Set breakpoints (click to the left of line numbers).
3. Run in "Debug" mode.

4. Step through code, inspect variables, etc.

4. Using Logging Instead of Print (for larger projects)

```
import logging

logging.basicConfig(level=logging.DEBUG)

def divide(a, b):

    logging.debug(f"Dividing {a} by {b}")

    return a / b

x=divide(10,0)
```

- `print(x)`Better than print for real applications
- Log levels: DEBUG, INFO, WARNING, ERROR, CRITICAL

5. Using `try/except` Blocks

Helpful for catching and understanding errors.

```
try:
    result = 10 / 0
except ZeroDivisionError as e:
    print("Error:", e)
```

Best Practice

Use a **debugger (like `pdb` or IDE tools)** when:

- The bug is hard to isolate
- You want to inspect variables live
- You want to step through execution

Use **`print()` or `logging`** when:

- You want quick visibility into what your code is doing
- You're in early development or prototyping

Debugging with `pdb` (Python's built-in debugger)

Let's say you want to debug this portion of your code:

```
my_stack = Stack()
my_stack.push(10)
my_stack.push(20)
print(f"Top element: {my_stack.peek()}")
print(f"Popped element: {my_stack.pop()}")
print(f"Stack size: {my_stack.size()}")
```

Step-by-step:

1. **Add `import pdb` at the top.**
2. **Insert `pdb.set_trace()` before the line you want to inspect.**

Here's your code with `pdb`:

```
import pdb

class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return len(self.items) == 0

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            print("Stack is empty")
            return None

    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            print("Stack is empty")
            return None

    def size(self):
        return len(self.items)

# Example usage:
my_stack = Stack()
my_stack.push(10)
my_stack.push(20)

pdb.set_trace()  # □ Debugger will stop here

print(f"Top element: {my_stack.peek()}")
print(f"Popped element: {my_stack.pop()}")
print(f"Stack size: {my_stack.size()}")
```

When you run this:

Type commands in the terminal when the program pauses:

- `l` — list source code
- `n` — next line
- `s` — step into method
- `p my_stack.items` — print stack contents
- `c` — continue running the code
- `q` — quit debugger

Debugging in VS Code (Visual Studio Code)

Setup for VS Code:

1. Open your Python file in VS Code.
2. On the left sidebar, click the **Run and Debug** icon ▶□.
3. Click **"Run and Debug"** > **Python File**.
4. Set breakpoints:
   - Click to the **left of the line numbers** to add a red dot (breakpoint).
5. Press **F5** to start debugging.

Useful VS Code Debug Tools:

- **Step Over (F10):** Runs the next line
- **Step Into (F11):** Jumps into method/function call
- **Variables panel:** See values in real time
- **Call stack panel:** See function call hierarchy
- **Watch panel:** Track specific variables

Example:

If you set a breakpoint at:

```
print(f"Top element: {my_stack.peek()}")
```

You can inspect `my_stack.items` in the **Variables** tab or just hover over `my_stack`.