1️⃣ Program 1 – sys.argv (Shortest & Raw)

Idea: Just read values directly from the list sys.argv.

File: calc_sysargv_simple.py

```
# calc_sysargv_simple.py
# Mini calculator using sys.argv
# Usage:
#   python calc_sysargv_simple.py <num1> <num2> <operation>
# Example:
#   python calc_sysargv_simple.py 10 5 add

import sys

# We need exactly 3 extra arguments: num1, num2, operation
if len(sys.argv) != 4:
   print("Usage: python calc_sysargv_simple.py <num1> <num2> <operation>")
   print("operation: add, sub, mul, div")
   sys.exit(1)

# Get values from the list (all are strings)
num1_str = sys.argv[1]
num2_str = sys.argv[2]
op = sys.argv[3]

# Convert numbers
try:
   num1 = float(num1_str)
   num2 = float(num2_str)
except ValueError:
   print("Error: num1 and num2 must be numbers.")
   sys.exit(1)

# Do the operation
if op == "add":
   result = num1 + num2
elif op == "sub":
   result = num1 - num2
elif op == "mul":
   result = num1 * num2
elif op == "div":
   if num2 == 0:
      print("Error: Cannot divide by zero.")
      sys.exit(1)
   result = num1 / num2
else:
   print("Error: operation must be: add, sub, mul, div")
```

```
    sys.exit(1)

print(f"[sys.argv] Result = {result}")
Run example:
python calc_sysargv_simple.py 10 5 add
```

2️⃣ Program 2 – getopt (Short Flags Like Unix Tools)

Idea: Use flags like -a, -b, -o instead of relying on position.

File: calc_getopt_simple.py

```
# calc_getopt_simple.py
# Mini calculator using getopt (short options)
# Usage:
#   python calc_getopt_simple.py -a <num1> -b <num2> -o <operation>
# Example:
#   python calc_getopt_simple.py -a 10 -b 5 -o mul

import sys
import getopt

def print_usage():
    print("Usage: python calc_getopt_simple.py -a <num1> -b <num2> -o <operation>")
    print("operation: add, sub, mul, div")

def main():
    num1 = None
    num2 = None
    op = None

    try:
        # "a:b:o:" = -a needs value, -b needs value, -o needs value
        opts, _ = getopt.getopt(sys.argv[1:], "a:b:o:h")
    except getopt.GetoptError as err:
        print("Error:", err)
        print_usage()
        sys.exit(1)

    for opt, val in opts:
        if opt == "-h":
            print_usage()
            sys.exit(0)
        elif opt == "-a":
            num1 = val
        elif opt == "-b":
            num2 = val
```

```
        elif opt == "-o":
            op = val

    # Check all required options are present
    if num1 is None or num2 is None or op is None:
        print("Error: -a, -b and -o are required.")
        print_usage()
        sys.exit(1)

    # Convert numbers
    try:
        num1 = float(num1)
        num2 = float(num2)
    except ValueError:
        print("Error: num1 and num2 must be numbers.")
        sys.exit(1)

    # Do the operation
    if op == "add":
        result = num1 + num2
    elif op == "sub":
        result = num1 - num2
    elif op == "mul":
        result = num1 * num2
    elif op == "div":
        if num2 == 0:
            print("Error: Cannot divide by zero.")
            sys.exit(1)
        result = num1 / num2
    else:
        print("Error: operation must be: add, sub, mul, div")
        sys.exit(1)

    print(f"[getopt] Result = {result}")

if __name__ == "__main__":
    main()
```

Run example:

```
python calc_getopt_simple.py -a 10 -b 5 -o mul
```

---

3 Program 3 – argparse (Modern & Clean)

Idea: Let argparse handle help, parsing, and type conversion.

File: calc_argparse_simple.py

```
# calc_argparse_simple.py
```

```
# Mini calculator using argparse (best for real projects)
# Usage:
#   python calc_argparse_simple.py -a 10 -b 5 -o add
#   python calc_argparse_simple.py --num1 10 --num2 5 --operation mul

import argparse

def main():
    # Create the parser
    parser = argparse.ArgumentParser(
        description="Mini calculator using argparse."
    )

    # Add arguments
    parser.add_argument("-a", "--num1", type=float, required=True, help="First number")
    parser.add_argument("-b", "--num2", type=float, required=True, help="Second
number")
    parser.add_argument(
        "-o", "--operation",
        choices=["add", "sub", "mul", "div"],
        required=True,
        help="Operation: add, sub, mul, div",
    )

    # Parse them
    args = parser.parse_args()

    # Do the operation
    if args.operation == "add":
        result = args.num1 + args.num2
    elif args.operation == "sub":
        result = args.num1 - args.num2
    elif args.operation == "mul":
        result = args.num1 * args.num2
    elif args.operation == "div":
        if args.num2 == 0:
            print("Error: Cannot divide by zero.")
            return
        result = args.num1 / args.num2

    print(f"[argparse] Result = {result}")

if __name__ == "__main__":
    main()
Run example:
python calc_argparse_simple.py -a 10 -b 5 -o add
```

4️⃣ How They Differ (Super Short Version)

- **sys.argv** → Just a **list of strings**. You do everything yourself.
- **getopt** → Adds **flags like -a, -b**, but you still handle most errors.
- **argparse** → Modern way. **Built-in --help, type checking, choices, nice errors.**

5️⃣ New, Shorter Video Script (Line-by-Line)

You can read this directly in your video.

🎬 Intro

1. "Hey everyone, welcome back to my channel."
2. "In this video, we'll learn three ways to handle command-line arguments in Python."
3. "We'll use one simple example: a mini calculator that works from the terminal."
4. "We'll start with sys.argv, then getopt, and finally argparse."
5. "I'll keep the code short and beginner friendly."

🧠 What are command-line arguments?

6. "When we run something like python script.py 10 5 add, the values 10 5 add are command-line arguments."
7. "They let us pass input to Python directly from the terminal without using input()."
8. "Now let's see three different ways to read them."

1️⃣ Part 1 – sys.argv

9. "First method: sys.argv."
10. "sys.argv is just a list of strings – everything we typed after python."
11. "In calc_sysargv_simple.py, I check if we have exactly three extra arguments: num1, num2, and operation."
12. "If the count is wrong, I print a usage message and exit."

13. “Then I read sys.argv[1], sys.argv[2], and sys.argv[3], and convert the first two to numbers.”
14. “I use simple if-elif to handle add, sub, mul, and div.”
15. “Finally, I print the result with a [sys.argv] label.”
16. “To run it: python calc_sysargv_simple.py 10 5 add.”
17. “This method is simple, but we have to write all the checks ourselves.”

2️⃣ Part 2 – getopt

18. “Second method: getopt, which feels more like Unix-style tools.”
19. “Instead of relying on position, we use flags like -a, -b, and -o.”
20. “In calc_getopt_simple.py, I define supported options using getopt.getopt.”
21. “If the user passes a wrong option, I show an error and a usage message.”
22. “I loop over the options: when I see -a, I set num1; -b sets num2; -o sets the operation.”
23. “Then I convert num1 and num2 to float and perform the calculation just like before.”
24. “I print the result with a [getopt] label.”
25. “To run it: python calc_getopt_simple.py -a 10 -b 5 -o mul.”
26. “This is cleaner than raw sys.argv, but we still write manual help and errors.”

3️⃣ Part 3 – argparse

27. “Third method: argparse, the modern and recommended way.”
28. “argparse automatically gives us --help, type conversion, and better error messages.”
29. “In calc_argparse_simple.py, I create a parser and add three arguments: --num1, --num2, and --operation.”
30. “For num1 and num2 I set type=float and required=True, so argparse converts and checks them for us.”
31. “For operation, I use choices=['add', 'sub', 'mul', 'div'] to restrict valid values.”
32. “Then I call parser.parse_args() and use the same if-elif logic for the operation.”

33. “Finally, I print the result with an [argparse] label.”
34. “To run it: python calc_argparse_simple.py -a 10 -b 5 -o add.”
35. “If I type --help, argparse shows a nice help message automatically.”

⚖️ Quick Comparison

36. “So, when should you use which one?”
37. “Use sys.argv for tiny scripts and quick experiments.”
38. “Use getopt if you like Unix-style flags but want to keep things low-level.”
39. “Use argparse for real tools, projects, and anything you’ll share with others.”
40. “All three solved the same calculator problem, but argparse gave the best user experience with the least manual work.”

✅ Outro

41. “That’s it for sys.argv, getopt, and argparse in Python.”
42. “I hope these short examples made it easy to understand the differences.”
43. “If you’re following my Python automation series, you can now make much better command-line tools.”
44. “Like the video, subscribe to the channel, and comment what topic you want next.”
45. “Thanks for watching, see you in the next video!”