📝 Lab 3 – Modify an Existing Script

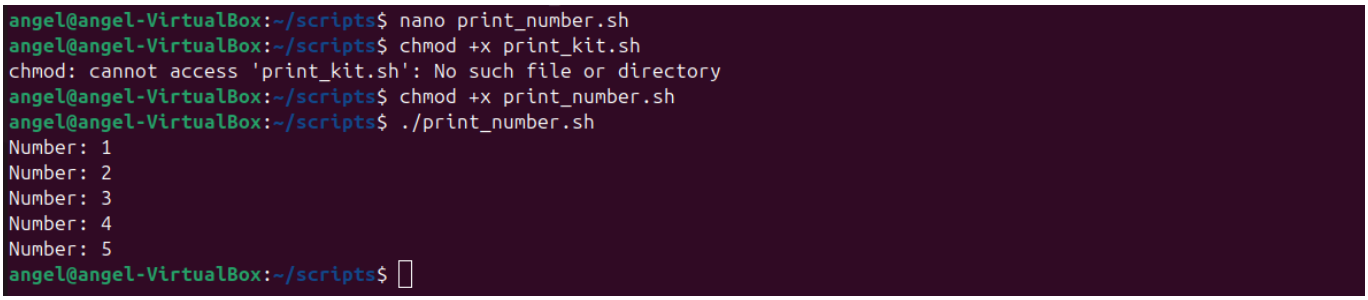
🎯 Objective

Enhance an existing Bash script (`print_numbers.sh`) to accept user input and add basic input validation.

🆚 Original vs Enhanced Script

Feature	`print_numbers.sh`	`enhanced_numbers.sh`
Fixed start & end	1 to 10	✅ User-defined start and end
Step size	Always 1	✅ User-defined step size
Input validation	✗ None	✅ Checks for valid and positive input
Flexibility	✗ Hardcoded	✅ Fully customizable

Output image of the codes :



🧪 Example Runs

```
#!/bin/bash
# enhanced_numbers.sh
# Print numbers from start to end with a custom step value

# === Input ===
read -p "Enter start value: " start
read -p "Enter end value: " end
read -p "Enter step value: " step

# === Validation ===
if ! [[ "$start" =~ ^-?[0-9]+$ && "$end" =~ ^-?[0-9]+$ && "$step" =~ ^[1-9]
[0-9]*$ ]]; then
  echo "✗ Invalid input. Start and end must be integers. Step must be a
positive integer."
  exit 1
fi

# === Output ===
```

```
echo "📌 Printing numbers from $start to $end (step: $step):"
for (( i=start; i<=end; i+=step ))
do
  echo "$i"
done
```

▶ Run 1 – Valid Input

```
$ ./enhanced_numbers.sh
Enter start value: 5
Enter end value: 15
Enter step value: 2
📌 Printing numbers from 5 to 15 (step: 2):
5
7
9
11
13
15
```

Run 2 - Invalid Run

```
$ ./enhanced_numbers.sh
Enter start value: 1
Enter end value: 10
Enter step value: 0
× Invalid input. Start and end must be integers. Step must be a positive
integer.
```

❓ Extra Questions

---

1. What is the difference between `$1`, `$@`, and `$#` in Bash?

Symbol	Description	Example
`$1`	First argument passed to the script	`./script.sh hello world` → `$1` = `hello`
`$@`	All arguments passed, treated as individual words	`for arg in "$@"; do ...` loops through each
`$#`	Total number of arguments passed	`./script.sh one two` → `$#` = `2`

---

2. What does `exit 1` mean in a script?

- `exit 1` means the script is **exiting with an error status**.
- In Bash scripting:

    - `exit 0` → Success
    - `exit 1` (or any non-zero) → Error or failure
- It is commonly used to:
    - Stop execution after an error
    - Signal to other programs or scripts that something went wrong

✅ Example:

```
if [ "$step" -le 0 ]; then
  echo "Invalid step value."
  exit 1  # Stops script with error
fi
``
```