Tasks – Print, sep, and end

Goal

You will create several small Python scripts that use `print()` in different ways. By the end you will be comfortable with basic output and with the `sep` and `end` parameters.

Prerequisites

- Python 3 installed (`python3 --version`).
- A text editor and a terminal/command line.

Task 1 – Simple print (welcome.py)

Objective: Run a script that prints a single greeting line.

Instructions:

1. Create a new file named `welcome.py` in this lab folder.
2. Add a shebang line at the top (optional but good practice): `#!/usr/bin/env python3`
3. Use `print()` to display the message: `Welcome to Python!`
4. Save the file and run: `python3 welcome.py`

Expected output:

```
Welcome to Python!
```

Deliverable: Script name: `welcome.py`

Task 2 – Print with multiple arguments (print_sep.py)

Objective: Use `print()` with several values and control how they are separated using `sep`.

Instructions:

1. Create a new file named `print_sep.py`.
2. Use a single `print()` call with three arguments: your first name, a comma, and your last name (as strings). Example: `"Alice"`, `","`, `"Smith"`.
3. Run the script. Notice the default behavior: Python inserts a space between each argument.
4. Add the parameter `sep=""` (empty string) to your `print()` call so that no space is added between the arguments. The output should look like: `Alice,Smith`
5. Then add a second `print()` in the same file that prints three words (e.g. `"Python"`, `"is"`, `"fun"`) with `sep="-"`. The line should print as: `Python-is-fun`

Expected output (adjust names/words as you like):

```
Alice,Smith
Python-is-fun
```

Deliverable: Script name: `print_sep.py`

Task 3 – Print without a newline (print_end.py)

Objective: Use the `end` parameter so that the next `print()` continues on the same line (or ends with something other than a newline).

Instructions:

1. Create a new file named `print_end.py`.
2. Use `print("Loading", end="")` so that nothing is printed after "Loading" (no newline).
3. Add a second line: `print("... Done!")`. When you run the script, the output should be one line: `Loading... Done!`
4. Add a third `print()` that prints two separate words on the same line by using `end=" "` (space) for the first print and a normal `print()` for the second. Example: first print `"Hello"` with `end=" "`, then print `"World"`.

Expected output:

```
Loading... Done!
Hello World
```

Deliverable: Script name: `print_end.py`

Task 4 – Combine sep and end (print_sep_end.py)

Objective: Use both `sep` and `end` in one or more `print()` calls to format a short line of data.

Instructions:

1. Create a new file named `print_sep_end.py`.
2. Print a "key: value" style line using one `print()` with two arguments (e.g. `"Name"` and `"Alice"`). Use `sep=": "` so it prints as `Name: Alice`. Use `end="\n"` explicitly (or omit it—it's the default) so the next output starts on a new line.
3. Print three numbers (e.g. `1`, `2`, `3`) in one `print()` with `sep=" | "` and `end=".\n"` so the line ends with a period and a newline. Output should look: `1 | 2 | 3.`

Expected output (values may vary):

```
Name: Alice
1 | 2 | 3.
```

Deliverable: Script name: `print_sep_end.py`

Task 5 – Status line (status_line.py)

Objective: Build a short script that prints a status line using `print()`, `sep`, and `end` together.

Instructions:

1. Create a new file named `status_line.py`.

2. Print a header line such as: `[STATUS]` followed by a short message. Use one `print()` with two arguments and `sep=" "` (e.g. `"[STATUS]"`, `"System ready."`).
3. Print a second line that looks like a simple key-value pair, e.g. `Time: 14:30`, using `sep=": "` in a single `print()`.
4. Optionally, use `end` so that a final "End of report." appears on the same line as the last character of the previous print (your choice), or on its own line.

Expected output (format similar to):

```
[STATUS] System ready.
Time: 14:30
End of report.
```

Deliverable: Script name: `status_line.py`

Completion

Congratulations! You have practiced:

- **print()** – basic output with one or more arguments.
- **sep** – controlling the string inserted *between* multiple arguments (default is a space).
- **end** – controlling what is printed *after* the last argument (default is a newline `\n`).

Next steps: Try a lab on variables or conditionals to use these output skills with data and logic.