Tasks – Comments in Python

Practice single-line comments (`#`) and multi-line comments (docstrings). Create each file, run it, and see how comments affect (or don't affect) output.

Run scripts with: `python3 script_name.py`

Task 1 – Single-line comment above code (`comment_above.py`)

- Create `comment_above.py` .
- Add a line that is only a comment: `# This script prints a greeting.`
- On the next line, add: `print("Hello")`
- Run the script. The comment should not appear in the output.

Expected output:

```
Hello
```

Task 2 – Inline comment (`comment_inline.py`)

- Create `comment_inline.py` .
- Write a line that does something and has a comment on the same line, e.g. `print("Hi") # say hi`
- Run the script. Only the effect of the code runs; the comment is ignored.

Expected output:

```
Hi
```

Task 3 – Multiple comment lines (`comment_block.py`)

- Create `comment_block.py` .
- Add 2–3 lines that are only comments (each starting with `#`), then a `print()` that runs.
- Run the script. Only the `print` output should appear.

Expected output (example):

```
Ready.
```

Task 4 – Commented-out code (`comment_out.py`)

- Create `comment_out.py` .
- Write a line that would print something, but put `#` at the start of that line so it does not run (e.g. `# print("skipped")`).
- On the next line, write an active `print("running")` .
- Run the script. Only the second line should produce output.

Expected output:

```
running
```

Task 5 – Docstring at top of file (`comment_docstring.py`)

- Create `comment_docstring.py` .
- As the first line of the file, add a docstring: three double quotes, a short description (e.g. `"""This script says done."""`), and closing `"""` .
- On the next line, add `print("Done")` .
- Run the script. The docstring is not printed; only `Done` appears.

Expected output:

```
Done
```

Task 6 – Multi-line docstring (`comment_multiline.py`)

- Create `comment_multiline.py` .
- Use a triple-quoted string that spans two or three lines (e.g. `"""Line one.` then `Line two."""`) at the top of the file. This acts as a multi-line comment or module docstring.
- Then add `print("OK")` .
- Run the script. Only `OK` should be printed.

Expected output:

```
OK
```

Task 7 – Comments and code together (`comment_mixed.py`)

- Create `comment_mixed.py` .
- Write a short script that: has one comment explaining what the script does, assigns a number to a variable, has an inline comment next to that line, and prints the variable. Use both “comment above” and “inline” style.

Expected output (example):

```
42
```

Task 8 – Uncomment to fix (`comment_uncomment.py`)

- Create `comment_uncomment.py` .
- Write a line that is commented out: `# print(2 + 2)` .
- Run the script; there should be no output.

- Then remove the `#` so the line becomes `print(2 + 2)`, save, and run again. The output should be `4`.

Expected output (after uncommenting):

```
4
```

Done

You've used: `#` for single-line and inline comments, commented-out code, and `"""..."""` (or `'''...'''`) for docstrings / multi-line comments. Comments are ignored when the script runs.