Tasks – Parameters in Python

Practice using **positional arguments**, **keyword arguments**, **default parameter values**, and the rules for parameter order. Create each file, run it, and check the output.

Run scripts with: `python3 script_name.py`

Task 1 – Two positional parameters (`param_two.py`)

- Create `param_two.py`.
- Define a function `describe(name, age)` that takes two parameters and prints e.g. `"Name: [name], Age: [age]"` using the two values.
- Call it with two arguments in order: `describe("Ali", 25)`.
- The first argument goes to `name`, the second to `age`.

Expected output:

```
Name: Ali, Age: 25
```

Task 2 – Positional order matters (`param_order.py`)

- Create `param_order.py`.
- Define a function `subtract(a, b)` that returns `a - b`.
- Call and print: `print(subtract(10, 3))` then `print(subtract(3, 10))`. The order of arguments determines which parameter gets which value.

Expected output:

```
7
-7
```

Task 3 – One parameter with a default value (`param_default_one.py`)

- Create `param_default_one.py` .
- Define a function `greet(name, greeting="Hello")` so that `greeting` has a default of `"Hello"` . In the body, print `greeting + ", " + name + "!"` .
- Call `greet("Bob")` with **one** argument. The second parameter uses its default.

Expected output:

```
Hello, Bob!
```

Task 4 – Override the default (`param_default_override.py`)

- Create `param_default_override.py` .
- Use the same idea: `def greet(name, greeting="Hello"):` and print the greeting and name.
- Call `greet("Bob")` (uses default) and then `greet("Bob", "Hi")` (overrides default). Print both outputs.

Expected output:

```
Hello, Bob!
Hi, Bob!
```

Task 5 – Keyword arguments (`param_keyword.py`)

- Create `param_keyword.py` .
- Define a function `info(name, age, city)` that prints e.g. `"name age city"` or `"Name: ... Age: ... City: ..."` using the three parameters.
- Call it using **keyword arguments**: `info(name="Alice", age=30, city="Tel Aviv")` . Then call again with the **same names but in a different order**: `info(city="London", name="Bob", age=22)` . Keyword order does not matter.

Expected output (example):

```
Name: Alice, Age: 30, City: Tel Aviv
Name: Bob, Age: 22, City: London
```

Task 6 – Mix positional and keyword (`param_mix.py`)

- Create `param_mix.py` .
- Define a function `format_msg(from_name, to_name, separator=" -> ")` that returns `from_name + separator + to_name` .
- Call with two positionals and the default: `format_msg("Alice", "Bob")` and print the result.
- Call with two positionals and a keyword to override: `format_msg("Alice", "Bob", separator=" | ")` and print. When mixing, put **positional arguments first**, then keyword arguments.

Expected output:

```
Alice -> Bob
Alice | Bob
```

Task 7 – Required before default in definition (`param_required_default.py`)

- Create `param_required_default.py` .
- Define `def multiply(a, b=2):` so that `a` is required and `b` defaults to `2` . Return `a * b` .
- Call and print: `print(multiply(5))` (uses b=2) and `print(multiply(5, 3))` (overrides b). In a function definition, **parameters with defaults must come after** parameters without defaults.

Expected output:

```
10
15
```

Task 8 – Multiple default parameters (`param_multi_default.py`)

- Create `param_multi_default.py`.
- Define `def greet(name, greeting="Hello", punctuation="!"):` and in the body print `greeting + ", " + name + punctuation`.
- Call and print: `greet("Sam")` (all defaults), then `greet("Sam", "Hi")` (override first default), then `greet("Sam", "Hey", "?")` (override both). Arguments are assigned to parameters **left to right** unless you use keyword names.

Expected output:

```
Hello, Sam!
Hi, Sam!
Hey, Sam?
```

Task 9 – Keyword to skip a default (`param_keyword_skip.py`)

- Create `param_keyword_skip.py`.
- Define `def connect(host, port=80, secure=False):` and in the body print e.g. `"host:port secure"` using the three parameters (e.g. `print(host, port, secure)`).
- Call with **one positional** and **one keyword** for the third parameter: `connect("example.com", secure=True)`. So `host="example.com"`, `port` stays 80, `secure` is True. Use a keyword argument to "skip" the second parameter and set the third.

Expected output (example):

```
example.com 80 True
```

Task 10 – Real-world style: required + optional (`param_real.py`)

- Create `param_real.py`.
- Define a function `log(message, level="INFO"):` that prints `"[level] message"` (e.g. `"[INFO] Server started"`).

- Call `log("Server started")` then `log("Error occurred", "ERROR")` then `log("Debug trace", level="DEBUG")`. Use a mix of positional and keyword so that the optional `level` can be omitted or set by name.

Expected output:

```
[INFO] Server started
[ERROR] Error occurred
[DEBUG] Debug trace
```

Done

You've used: **positional arguments** (order matters), **keyword arguments** (name=value, order doesn't matter), **default parameter values**, overriding defaults by passing arguments, **mixing** positional and keyword in a call, and the rule that **required parameters come before parameters with defaults** in the function definition.