

Functions: Arguments, Returns & Good Design

Overview: what you'll produce

By the end of this lab you will have a single folder, `code/labs/lab-04/`, containing four working Python scripts, each one runnable on its own from the terminal:

- `functions_basics.py` - defines and calls functions with positional, keyword and default arguments
- `args_practice.py` - uses `*args` to accept any number of arguments
- `returns.py` - explores return values, including returning more than one value at once
- `stats_toolkit.py` - a small mini-project: a toolkit of well-designed statistics functions

Nothing here is handed in — it's practice, so work at your own pace and don't worry about getting every exercise perfect. Some students will get through this faster than others depending on how much Python experience they already have, and that's expected. If you don't finish everything in the three hours, use your independent study time to complete the remaining exercises.

Before you start

This lab builds on Labs 1 to 3 — you should already have a `code/labs/` folder from last time. Save every script inside a new folder called `code/labs/lab-04/` to keep things organised. Check that:

- ☐ You completed Labs 1 to 3, and their folders still exist inside `code/labs/`
- ☐ You've written and called your own simple functions before, for example `main()` in earlier labs

Lab objectives

- Call functions using positional arguments, keyword arguments, and default parameters
- Use `*args` to accept any number of arguments into a function
- Understand the difference between a function that `returns` a value and one that just `print()`s it
- Return more than one value from a function at once, and unpack the result
- Write single-purpose functions with purposeful names, docstrings, and no reliance on global variables

Timing guide

Time	Exercise	Focus
0:00 – 0:10	Setup	Create working folder

0:10 – 0:40	Exercise 1	Positional, keyword and default arguments
0:40 – 1:10	Exercise 2	*args
1:10 – 1:45	Exercise 3	Return values
1:45 – 1:55	Break	—
1:55 – 2:50	Exercise 4	Mini project: Stats Toolkit
2:50 – 3:00	Wrap-up	Self-check

A note on functions

A function is a named, reusable block of code that does one job. You've already written and called functions in every lab so far — this lab looks more closely at what goes in (arguments) and what comes out (return values), and at what separates a function that's well designed from one that just happens to work. Writing good functions isn't only about correct output — two programs can behave identically while one is minimal, clean and easy to reuse, and the other is a tangle that's hard to test, fix, or hand to someone else.

Exercise 1 — Positional, keyword and default arguments (30 min)

Create a new file called `functions_basics.py` for this exercise.

A note on syntax hints: where a step includes a “Syntax hint,” it shows the general shape of the code, using placeholder names like `function_name` and `param1` — it is not code to type in as-is. Treat it as a pattern to follow, writing it with your own function and variable names.

1. Write a function called `display_book` that takes two parameters, `title` and `author`, and prints them together in a single line, e.g. `"The Hobbit – J.R.R. Tolkien"`. Call it twice with two different books, matched by position. Syntax hint:

```
def function_name(param1, param2):
    ...
```

2. Call `display_book` again, but this time using keyword arguments — `author=`, `title=` — written in the opposite order from how the parameters were defined. The result is identical; only how you write the call has changed.

3. Add a third parameter, `year`, with a default value of `None` (for when the year of publication isn't known). Call the function once relying on the default, and once overriding it with a keyword argument. Syntax hint:

```
def function_name(param1, param2, param3=default_value):  
    ...
```

Checkpoint

`functions_basics.py` should define a function (`display_book`) with at least one default parameter, and call it using positional arguments, keyword arguments, and by relying on the default at least once.

Exercise 2 — `*args`: any number of arguments (30 min)

Create a new file called `args_practice.py` for this exercise.

1. Write a function `total_of(*numbers)` that adds up however many numbers it's given, using a for loop and a running total (the same pattern you used in Lab 3), and returns the result. Syntax hint for accepting any number of arguments:

```
def function_name(*args):  
    ...
```

2. Call it three times: once with two numbers, once with five, and once with none at all. Print each result.
3. Inside `total_of`, print `type(numbers)` before the loop runs — the collected arguments arrive bundled together as a tuple, whatever you call the parameter.
4. Write a second function, `describe_fridge(*items)`, that uses a for loop with `range(len(items))` to print each item labelled with its position, e.g. "1: apples".

Checkpoint

`args_practice.py` should define and call at least one function using `*args`, call it with different numbers of arguments (including zero), and show that the collected arguments arrive as a tuple.

— 10 minute break —

Exercise 3 — Return values (35 min)

Create a new file called `returns.py` for this exercise.

1. Write a function that `print()`s a doubled number instead of returning it, then try to capture its result in a variable and print that variable. Read what you get.

```
def double_bad(n):  
    print(n * 2)
```

```
result = double_bad(5)  
print(result)
```

2. Notice `result` printed as `None` — printing inside a function shows something on screen, but hands nothing back to the code that called it. Fix the function so it `returns` the doubled value instead, then print the captured result.

```
def double_good(n):  
    return n * 2
```

```
result = double_good(5)  
print(result)
```

3. Write `min_max(numbers)` that returns both the lowest and highest number in a list as `return lowest, highest` — this packs them into a single tuple behind the scenes. You already know how to loop through a list while keeping track of a running best value, from the Quiz Score Tracker in Lab 3 — apply the same idea here, once for the lowest and once for the highest.
4. Call it and unpack the result into two separate variables. Syntax hint: `lo, hi = min_max(your_list)`. Print both.
5. Deliberately unpack into the wrong number of variables, for example `a, b, c = min_max([1, 2, 3])`, run it, and read the `ValueError` — the number of names on the left must match the number of values returned. Then fix it back.

Checkpoint

`returns.py` should show the difference between printing and returning (including a variable that ends up holding `None` by mistake), and a function that returns two values at once as a tuple, correctly unpacked.

Exercise 4 — Mini project: Stats Toolkit (55 min)

Bring together everything from this lab into one program, `stats_toolkit.py`, that builds a small toolkit of well-designed statistics functions.

Requirements:

- `calculate_average(numbers)` - returns the average of a list of numbers, with a docstring
- `find_min_max(numbers)` - returns the lowest and highest number as a list, with a docstring
- `count_above(numbers, threshold=50)` - returns how many numbers are at or above a threshold, with a default of 50, with a docstring

- Each function does one job, takes the list as a parameter (no globals), and returns its result rather than printing it
- Organise the program's logic inside a `main()` function that calls all three on a sample list and prints a short summary report

Example run:

Numbers: [42, 67, 88, 23, 91, 55, 30]

Average: 56.6

Lowest: 23, Highest: 91

Count at or above 50: 4

Checkpoint

Running `stats_toolkit.py` should show three single-purpose functions, each with a docstring, each returning rather than printing, with `count_above` using its default threshold at least once, and a `main()` that calls all three and prints a summary.

Self-check

Before you finish up, look back over what you've written — this isn't handed in, but it's worth confirming you've hit the key ideas:

- ☐ `code/labs/lab-04/` contains: `functions_basics.py`, `args_practice.py`, `returns.py`, `stats_toolkit.py`
- ☐ Every script runs from the terminal without errors: `python <file>.py`
- ☐ At least one function uses `*args` and at least one uses a default parameter
- ☐ At least one function returns more than one value, correctly unpacked at the call site
- ☐ `stats_toolkit.py`'s three functions are each single-purpose, documented, and return rather than print, called from `main()`