Python Programming
Unit 02 – Lecture 04 Notes
Functions Basics and Parameters

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

Functions let you name a block of logic and reuse it. This is one of the most important habits for writing clean programs: instead of copying code, you call a function.

This lecture covers:

- function definition syntax (`def`, parameters, `return`),
- docstrings and why documentation matters,
- positional vs keyword arguments,
- default arguments (and the common mutable-default trap),
- variable-length arguments (`*args`, `**kwargs`), and
- local vs global scope.

2 Why Functions?

Functions improve code quality in multiple ways:

- **Avoid repetition:** write logic once, reuse many times.
- **Readability:** a good name summarizes the intent.
- **Testing:** small functions are easier to test than long scripts.
- **Abstraction:** callers do not need to know internal details.

3 Function Syntax and Behavior

3.1 Basic Definition

```python
def add(a, b):
    return a + b

result = add(3, 4)
```

Key parts:

- `def` starts a function definition.
- Parameters (`a`, `b`) are inputs.
- `return` sends a value back to the caller.

3.2 `return` vs `print`

A common beginner confusion is mixing output with return values.

- `return` gives a value back to the caller for further use.
- `print` only displays text; it does not produce a reusable value.

3.3 Docstrings

A docstring is the first string in a function body. It documents the purpose, inputs, and output.

```python
def area(r):
    """Return the area of a circle with radius r."""
    return 3.14 * r * r
```

Docstrings can be viewed with `help(area)` or via `area.__doc__`.

4 Parameters and Arguments

4.1 Parameters vs Arguments

- **Parameters** appear in the function definition.
- **Arguments** are the actual values passed during a call.

Example:

```python
def greet(name, msg): # parameters
    return f"{msg}, {name}"

print(greet("Asha", "Hello")) # arguments
```

4.2 Positional and Keyword Arguments

```python
def power(base, exp):
    return base ** exp

power(2, 3) # positional
power(base=2, exp=3) # keyword
power(exp=3, base=2) # keyword order can change
```

Keyword arguments improve readability when a function has many parameters.

5 Default Arguments

Defaults make a function flexible:

```python
def greet(name, msg="Hello"):
    return f"{msg}, {name}"
```

Important detail: default values are evaluated once, when the function is defined (not every time it is called).

5.1 Pitfall: Mutable Default Arguments

This is one of the most common Python interview and exam traps.

```python
def add_item(x, items=[]):
    items.append(x)
    return items

print(add_item(1)) # [1]
print(add_item(2)) # [1, 2] (surprise!)
```

Why it happens: the default list `[]` is created once and reused.

5.2 Safe Pattern for Defaults

Use `None` and create a new list inside the function:

```python
def add_item_safe(x, items=None):
    if items is None:
        items = []
    items.append(x)
    return items
```

6 Variable-Length Arguments

Sometimes you do not know how many inputs the caller will provide.

6.1 `*args` (Extra Positional Arguments)

```python
def total(*nums):
    return sum(nums)

print(total(10, 20, 30))
```

Inside the function, `nums` is a tuple of all extra positional arguments.

6.2 `**kwargs` (Extra Keyword Arguments)

```python
def describe(**info):
    return info

print(describe(name="Asha", age=18))
```

Inside the function, `info` is a dictionary of keyword arguments.

6.3 Worked Example: Average of Any Number of Scores

```python
def avg(*scores):
    if not scores:
        return 0.0
    return sum(scores) / len(scores)
```

The empty-input check avoids division by zero.

7 Scope: Local vs Global

By default, variables created inside a function are **local**.

```python
x = 5

def foo():
    x = 2 # local
    return x

print(foo(), x)
```

Using global variables is possible (with the `global` keyword), but it makes code harder to reason about. Prefer returning values instead.

8 Demo Walkthrough: Grade Calculator Function

Script: `demo/grade_calculator.py`

8.1 What the Demo Is Teaching

- Use a function to keep grading logic in one place.
- Use parameters/defaults so the grading scheme can be reused.
- Return a grade value (instead of only printing).

8.2 Suggested Run Steps

```
python demo/grade_calculator.py
```

Extension idea: allow custom grade boundaries (e.g., a different university policy).

9 Quiz and Solutions

9.1 Checkpoint 1 (Mutable Defaults)

Question. What happens if you use a mutable default like `def f(items=[])`?

Solution. The default object is created once and reused across calls. So changes to `items` in one call affect the default in later calls.

9.2 Checkpoint 2 (Local Scope)

Question. Predict the output:

```python
x = 5

def foo():
    x = 2
    return x

print(foo(), x)
```

Solution. Output is `2 5`. The `x` inside `foo` shadows the global `x` but does not change it.

9.3 Think-Pair-Share (Keyword Arguments)

Prompt. When do keyword arguments improve readability?

Sample response. When a function has multiple parameters of the same type or meaning, keywords prevent confusion. For example, `resize(width=640, height=480)` is clearer than `resize(640, 480)`.

9.4 Exit Question (Signature for Average)

Question. Write a function signature that accepts any number of scores and returns their average.

Solution.

```python
def avg(*scores):
    if not scores:
        return 0.0
    return sum(scores) / len(scores)
```

10 Practice Exercises (With Solutions)

10.1 Exercise 1: Safe Append Helper

Task. Write a function `push(x, items=None)` that appends `x` to a list, but does not use a mutable default.

Solution.

```python
def push(x, items=None):
    if items is None:
        items = []
    items.append(x)
    return items
```

10.2 Exercise 2: Named Formatting Function

Task. Write `format_name(first, last, title="")` that returns a formatted full name. Prefer keyword arguments for optional parts.

Solution.

```python
def format_name(first, last, title=""):
    title = title.strip()
    if title:
        return f"{title} {first} {last}"
    return f"{first} {last}"
```

10.3 Exercise 3: Return vs Print

Task. Write a function `is_even(n)` that returns `True` or `False`. Do not print inside the function.

Solution.

```python
def is_even(n):
    return n % 2 == 0
```

11 Further Reading

- `https://docs.python.org/3/tutorial/controlflow.html#defining-functions`