1) What is a function? (Simple idea)

A function is a named block of code that does one specific job.
Think of a function like a little machine or recipe. You give it inputs, it does a task, and then it might give you back a result.
Why use functions?

- To avoid repeating the same code.
- To make code easier to read and understand.
- To allow testing one small piece at a time.

2) How to create (define) a function

In Python we use the keyword def, the function name, parentheses (), a colon :, and an indented block of code below. That indented block is the function body.

Example:

python Copy code

```python
def say_hello():
    print("Hello!")
```

- def tells Python "I am defining a function".
- say_hello is the function name. Use descriptive names so the purpose is clear.
- The parentheses are required even if empty.
- The colon : signals the start of the function body.
- Everything inside the function must be indented the same amount (usually 4 spaces).

Important: **defining** a function does not run it. It only tells Python what the function does.

3) How to run (call) a function

To run the function you use its name followed by parentheses.

python Copy code

```python
say_hello()   # This runs the code inside the function
```

If you define but never call it, nothing happens.

4) Functions with inputs: parameters and arguments

A function often needs data to work on. We give names to the expected inputs in the function definition. Those names are called *parameters*.

Example:

python Copy code

```
def add(x, y):
    print(x + y)
```

- `x` and `y` are parameters.
- When we call the function, the actual values we pass are called *arguments*:

python Copy code

```
add(4, 5)   # 4 and 5 are arguments
```

Analogy: parameters are like labeled ingredient spaces in a recipe: "add 1 cup of flour" where "cup of flour" is supplied when you cook.

5) Returning values vs printing

Two common things a function can do with results:

- **Print** something directly inside the function.
- **Return** a value to the code that called the function.

Example that prints:

```python
def add_and_print(a, b):
    print(a + b)

add_and_print(2, 3)  # prints 5
```

Example that returns:

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

result = add(2, 3)   # result now holds 5
print(result)        # we decide when or if to print
```

Why prefer return most of the time?

- Returning lets the caller decide what to do next. You can store the value, print it, send it to another function, or save it to a file. Printing inside a function makes it harder to reuse the result.

6) Functions can return multiple values

A function can give back more than one value using commas. Python returns a tuple, and you can unpack it into variables.

Example:

```python
def add_and_sub(a, b):
    sum_value = a + b
    diff = a - b
    return sum_value, diff

s, d = add_and_sub(7, 2)
print(s, d)   # prints: 9 5
```

7) Function names and parameter names

- Use clear, descriptive names: calculate_tax is better than ct.

- Use lowercase and underscores for readability: calculate_total.
- Parameter names should show what they represent: price, quantity, name.

8) Indentation and scope (where variables live)

Indentation matters. Everything indented under def is inside that function.

Scope:

- Variables created inside a function exist only inside that function. They are local variables.
- Variables created outside are global to that file and not directly changed by the function unless passed and returned.

Example:

python Copy code

```python
def example():
    inside_var = 10   # inside_var only exists inside example()

example()
# trying to print inside_var here causes an error
```

If you want data out of a function, have it return that data.

9) Default values for parameters (optional inputs)

You can give parameters default values so the caller can omit them.

python Copy code

```python
def greet(name="friend"):
    print("Hello", name)

greet("Alice")  # Hello Alice
greet()         # Hello friend
```

10) Good and bad practices

Good:

- Small functions that do one clear job.
- Meaningful names.
- Prefer return over printing when building logic.

- Add short comments or a docstring describing what the function does.

Bad:

- One very long function that mixes many tasks.
- Using unclear names like f or x when a better name exists.
- Relying on global variables instead of passing and returning data.

11) Common beginner mistakes and how to fix them

1. Defining function but forgetting to call it.
    - Fix: add function_name() where you want it to run.
2. Wrong indentation.
    - Fix: make sure all code inside the function is indented the same number of spaces.
3. Using a variable from inside a function outside it.
    - Fix: return the value and store it in a variable outside.
4. Mixing up parameters and arguments.
    - Remember: parameters are names used in the function definition. Arguments are the actual values you pass when calling.
5. Printing instead of returning.
    - If you need to reuse the value, return it. Print only when you want to show it to the user.

12) Short reference table

Task	Syntax example
Define a function	def name(): then indented body
Call a function	name()
Parameters	def add(x, y):
Return value	return x + y
Multiple returns	return a, b
Default param	def greet(name="friend"):

Docstring	"""Describe function.""" below def line