

# Learning Python Functions

lets learn python  
start with function

Excellent! Let's start learning **functions** in Python in a very clear and structured way.

## 🌟 1 What is a Function?

A **function** is a block of reusable code that performs a specific task.

### Benefits:

- ✓ Avoid repeating code
- ✓ Make code easier to read and maintain
- ✓ Break big problems into smaller ones

## 🌟 2 Basic Structure

In Python, you define a function using the `def` keyword.

python

```
def function_name(parameters):  
    # Code block  
    return value
```

- ✓ `def` – tells Python you are defining a function
- ✓ `function_name` – the name you choose
- ✓ `parameters` – inputs to your function (optional)
- ✓ `return` – value to send back (optional)

## 🌟 3 Example 1: A Simple Function (No Parameters)

python

```
def say_hello():  
    print("Hello, World!")  
  
say_hello()
```

✓ Output:

Hello, World!

---

## 🌟 4 Example 2: Function with Parameters

python

```
def greet(name):  
    print("Hello,", name)  
  
greet("Alice")
```

✓ Output:

Hello, Alice

---

## 🌟 5 Example 3: Function with Return Value

python

```
def add(a, b):  
    return a + b  
  
result = add(5, 3)  
print(result)
```

✓ Output:

8

## 🌟 6 Default Parameters

You can set a default value:

python

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

```
greet()      # Hello, Guest
```

```
greet("Bob") # Hello, Bob
```

## 🌟 7 Keyword Arguments

You can specify arguments by name:

python

```
def introduce(name, age):
    print(f"My name is {name} and I'm {age} years old.")
```

```
introduce(age=25, name="Charlie")
```

✓ Output:

vbnet

My name is Charlie and I'm 25 years old.

## 🌟 8 \*args and \*\*kwargs

✓ `*args` – multiple positional arguments

✓ `**kwargs` – multiple keyword arguments

python

```
def show_args(*args):  
    print(args)  
  
show_args(1, 2, 3)
```

✓ Output:

scss

(1, 2, 3)

python

```
def show_kwargs(**kwargs):  
    print(kwargs)  
  
show_kwargs(name="Alice", age=30)
```

✓ Output:

bash

{'name': 'Alice', 'age': 30}

---

## 🌟 9 Docstrings (Documentation Strings)

Describe what your function does:

python

```
def add(a, b):  
    """Returns the sum of a and b."""  
    return a + b
```

```
help(add)
```

✓ Output:

```
sql
```

```
Help on function add in module __main__:
```

```
add(a, b)
```

```
  Returns the sum of a and b.
```

## 🌟 10 Best Practices

- ✓ Use descriptive names
- ✓ Write docstrings
- ✓ Keep them small and focused

## ✓ Quick Practice Exercise

✏ Write a function called `multiply` that takes two numbers and returns their product.

| If you want, you can write your solution here and I'll check it for you!

Or tell me if you want **more theory** or **more exercises** next!

```
def mul (a, b):  
    return a*b  
mul(8,7)
```

✓ Perfect! Your function works exactly as expected.

Here's a breakdown of your code:

```
python
```

```
def mul(a, b):    # Defines a function 'mul' that takes two arguments a and b
    return a * b  # Returns the product of a and b

mul(8, 7)        # Calls the function with 8 and 7
```

👉 However, since you're not using `print()`, the result will not be shown unless you're running this in an interactive shell like Jupyter Notebook or Python shell.

✅ **To see the output in a script or most environments, do this:**

```
python

print(mul(8, 7))
```

✅ **Output:**

```
56
```

Would you like to:

1. Learn about **function scope** and variables inside functions?
2. Practice with more **function exercises**?
3. Learn about **lambda functions** (short, anonymous functions)?
4. Move to the next topic in Python (like loops, lists, or conditionals)?

Let me know what you'd prefer!

Learn about function scope and variables inside functions?

Excellent choice! Let's talk about **function scope** and **variables inside functions** in Python.

I'll explain simply, with examples you can try.

## 🌟 1 What is Scope?

**Scope** is the region of your code where a variable is *accessible*.

In Python, there are two main types:

- ✅ **Global scope** – accessible everywhere in the file
  - ✅ **Local scope** – accessible only inside a function
- 

## 🌟 2 Example of Local Scope

Variables defined **inside** a function exist only there.

```
python
```

```
def greet():  
    message = "Hello, World!"  
    print(message)  
  
greet()
```

✅ Output:

```
Hello, World!
```

But this will give an error:

```
python
```

```
print(message) # ERROR! message is not defined outside the function
```

✅ Error:

```
pgsql
```

```
NameError: name 'message' is not defined
```

👉 `message` is *local* to `greet()`.

---

## ★ 3 Example of Global Scope

A variable defined **outside** any function is *global*:

```
python

name = "Alice" # Global variable

def greet():
    print("Hello,", name)

greet()
```

✓ Output:

```
Hello, Alice
```

✓ `name` is accessible inside the function because it's global.

---

## ★ 4 Shadowing

If you define a variable inside a function with the same name as a global one, the local version is used:

```
python

name = "Alice"

def greet():
    name = "Bob" # This is local!
    print("Hello,", name)

greet()
print(name)
```

✓ Output:



```
Hello, Bob
Alice
```

✓ Outside the function, `name` is still "Alice"!

## 🌟 5 The `global` Keyword

If you want to **modify** a global variable inside a function:

```
python

counter = 0

def increment():
    global counter
    counter += 1

increment()
print(counter)
```

✓ Output:

```
1
```

✓ Without `global`, you cannot assign to a global variable inside a function.

## 🌟 6 Nested Functions and Enclosing Scope

Functions can be defined *inside* other functions:

```
python

def outer():
    msg = "Hello"

    def inner():
```

```
print(msg)
```

```
inner()
```

```
outer()
```

✓ Output:

```
nginx
```

```
Hello
```

✓ `inner()` can access `msg` from `outer()`.

This is called **enclosing scope**.

## 🌟 7 LEGB Rule

Python uses **LEGB** to resolve variable names:

- **L**: Local — inside the current function
- **E**: Enclosing — in any outer functions
- **G**: Global — top-level of the module/script
- **B**: Built-in — Python's own names (like `len`, `sum`)

✓ Python looks for variables in this order!

## ✓ Quick Practice Exercise

✎ Define a global variable called `count` with value 5.

✎ Write a function `increase()` that adds 1 to `count`.

Try it yourself, or tell me if you want the **solution**!

If you want, just say:

✓ "Give me the solution"

Or tell me:

✓ "More on scope"

✓ "Next topic"

✓ "More exercises"

I'm here to help you learn step by step!