Defining and Calling Functions

Introduction

In the previous lesson, we learned that a **function** is a reusable block of code that performs a specific task.

But knowing what a function is is only the beginning.

To actually use a function in Python, we need to understand two basic actions:

1. **Defining a function**
2. **Calling a function**

These are different things.

When you **define** a function, you create and describe what it should do.

When you **call** a function, you tell Python to execute that code.

Think of it like creating a recipe:

```
Define the function
        ↓
Create the instructions
        ↓
Call the function
        ↓
Python follows the instructions
```

---

1. Defining a Function

Python uses the `def` keyword to define a function.

Basic syntax:

```
def function_name():
    # function body
```

Example:

```
def greet():
    print("Hello!")
```

Here we have created a function named `greet`.

But nothing is printed yet.

Why?

Because we only **defined** the function.

We haven't called it.

---

2. Understanding the Function Definition

Look carefully at this:

```
def greet():
    print("Hello!")
```

Each part has a purpose.

```
def
 ↓
Tells Python that we are defining a function
```

```
greet
  ↓
The name of the function
```

```
()
 ↓
Where parameters can be placed
```

```
:
 ↓
Marks the beginning of the function body
```

```
print("Hello!")
```

This is the code that belongs to the function.

---

3. The Function Body

The code inside a function is called the **function body**.

Example:

```
def greet():
    print("Hello!")
    print("Welcome to Python!")
```

Both `print()` statements belong to the function because they are indented.

The indentation is important.

Correct:

```
def greet():
    print("Hello!")
    print("Welcome!")
```

Incorrect:

```
def greet():
print("Hello!")
```

Python expects the function body to be indented.

---

4. Defining Does Not Mean Running

This is one of the most important things to understand.

Consider:

```
def greet():
    print("Hello!")
```

You might expect:

```
Hello!
```

But nothing appears.

Why?

Because Python has only created the function.

The function is waiting to be called.

Think of it like creating a button.

```
Define function
      ↓
Button exists
      ↓
Press button
      ↓
Function runs
```

---

5. Calling a Function

To execute a function, we **call** it.

Syntax:

```
function_name()
```

Example:

```
def greet():
    print("Hello!")


greet()
```

Output:

```
Hello!
```

The line:

```
greet()
```

is the function call.

---

6. Definition vs Call

Compare these two lines:

```
def greet():
    print("Hello!")
```

and:

```
greet()
```

The first one **defines** the function.

The second one **calls** the function.

Remember:

```
def greet():
    ↓
Create the function


greet()
    ↓
Run the function
```

---

7. A Function Can Be Called Multiple Times

One of the biggest benefits of functions is that we can reuse them.

Example:

```
def greet():
    print("Hello!")


greet()
greet()
greet()
```

Output:

```
Hello!
Hello!
Hello!
```

We wrote the function only once.

But we called it three times.

---

8. Why Is Reusability Important?

Imagine you need to print a welcome message in 20 different places.

Without a function:

```
print("Welcome to haas.dev!")
print("Welcome to haas.dev!")
print("Welcome to haas.dev!")
# repeated many times
```

With a function:

```
def welcome():
    print("Welcome to haas.dev!")


welcome()
welcome()
welcome()
```

The function contains the logic once.

You can reuse it whenever necessary.

---

9. Function Names

A function should have a meaningful name.

Good examples:

```
calculate_total()
show_profile()
get_username()
send_email()
check_password()
display_menu()
```

Poor examples:

```
abc()
thing()
do_it()
function1()
```

A good function name should give you an idea of what the function does.

---

10. Python Function Naming Convention

Python commonly uses **snake_case** for function names.

That means:

- lowercase letters
- words separated using `_`

Examples:

```
calculate_total()
get_user_name()
check_password()
display_result()
calculate_average()
```

Avoid:

```
CalculateTotal()
GetUserName()
calculateTotal()
```

For normal Python functions, prefer:

```
calculate_total()
```

---

11. Functions With Parameters

A function can receive information when it is called.

Example:

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

Here:

```
name
```

is a **parameter**.

Now call the function:

```
greet("Hafsa")
```

Output:

```
Hello Hafsa
```

The value `"Hafsa"` is passed into the function.

---

12. Calling the Same Function With Different Values

The same function can work with different arguments.

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


greet("Hafsa")
greet("Asim")
greet("Ali")
```

Output:

```
Hello Hafsa
Hello Asim
Hello Ali
```

The function doesn't need to be rewritten for every person.

---

13. Multiple Parameters

A function can accept multiple parameters.

Example:

```
def add(a, b):
    print(a + b)
```

Call it:

```
add(10, 20)
```

Output:

```
30
```

Another call:

```
add(50, 25)
```

Output:

```
75
```

---

14. Function Call Flow

Let's understand what Python does when we call a function.

Code:

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


greet("Hafsa")
```

Python first reads the function definition.

Then it reaches:

```
greet("Hafsa")
```

Python enters the function.

```
greet("Hafsa")
       ↓
name = "Hafsa"
       ↓
print("Hello", name)
       ↓
Hello Hafsa
```

After the function finishes, Python continues with the code after the function call.

---

15. Example With Code After the Function Call

```
def greet():
    print("Hello!")


print("Before")

greet()

print("After")
```

Output:

```
Before
Hello!
After
```

The function runs exactly where it is called.

---

16. Defining a Function Before Calling It

Normally, define a function before the point where you call it.

Correct:

```
def greet():
    print("Hello!")


greet()
```

This works because Python has already executed the function definition before reaching the call.

---

17. Calling Before Defining

Consider:

```
greet()

def greet():
    print("Hello!")
```

This causes an error because Python reaches:

```
greet()
```

before the function has been defined.

For beginners, follow this simple rule:

```
Define first.
Call afterward.
```

---

18. Functions With No Parameters

A function doesn't always need input.

Example:

```
def show_menu():
    print("1. Start")
    print("2. Settings")
    print("3. Exit")
```

Call it:

```
show_menu()
```

Output:

```
1. Start
2. Settings
3. Exit
```

---

19. Functions With One Parameter

Example:

```
def square(number):
    print(number * number)
```

Call:

```
square(5)
```

Output:

```
25
```

Another call:

```
square(10)
```

Output:

```
100
```

---

20. Functions With Multiple Parameters

Example:

```
def introduce(name, age):
    print("Name:", name)
    print("Age:", age)
```

Call:

```
introduce("Hafsa", 25)
```

Output:

```
Name: Hafsa
Age: 25
```

The arguments are matched with parameters in order:

```
name → "Hafsa"
age  → 25
```

---

21. Argument Order Matters

Consider:

```
def introduce(name, age):
    print(name)
    print(age)
```

Now:

```
introduce("Hafsa", 25)
```

means:

```
name = "Hafsa"
age = 25
```

But:

```
introduce(25, "Hafsa")
```

means:

```
name = 25
age = "Hafsa"
```

Python passes positional arguments according to their order.

---

22. Returning a Value

A function can also send a value back using `return`.

Example:

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

Call it:

```
result = add(10, 20)

print(result)
```

Output:

```
30
```

The function call produces a value that can be stored in a variable.

---

23. Calling a Function Inside Another Function

Functions can call other functions.

Example:

```
def greet():
    print("Hello!")


def start():
    greet()
    print("Program started")


start()
```

Output:

```
Hello!
Program started
```

The flow is:

```
start()
  ↓
greet()
  ↓
Hello!
  ↓
Program started
```

This becomes very useful in larger programs.

---

24. Functions as Building Blocks

A large program can be divided into smaller functions.

For example, a student management program might have:

```
def add_student():
    pass


def remove_student():
    pass


def search_student():
    pass


def display_students():
    pass
```

Each function handles a different task.

The main program can call them when needed.

This makes the program easier to organize.

---

25. Practical Example: Calculator

Let's create a simple calculator using functions.

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


def subtract(a, b):
    return a - b


def multiply(a, b):
    return a * b


def divide(a, b):
    return a / b
```

Now call the functions:

```
print(add(10, 5))
print(subtract(10, 5))
print(multiply(10, 5))
print(divide(10, 5))
```

Output:

```
15
5
50
2.0
```

Each function has one clear responsibility.

---

26. Practical Example: Student Result

Suppose we want to calculate a student's total marks.

```
def calculate_total(math, english, computer):
    return math + english + computer
```

Call it:

```
total = calculate_total(80, 75, 90)

print("Total:", total)
```

Output:

```
Total: 245
```

The function can be reused:

```
student1 = calculate_total(80, 75, 90)
student2 = calculate_total(70, 85, 88)

print(student1)
print(student2)
```

---

27. Function Execution Order

Consider this program:

```
def first():
    print("First")


def second():
    print("Second")


print("Start")

second()

first()

print("End")
```

Output:

```
Start
Second
First
End
```

Python doesn't automatically execute functions when they are defined.

It executes them when they are called.

---

28. Common Beginner Mistakes

Mistake 1: Defining but Never Calling

```
def greet():
    print("Hello!")
```

Nothing appears.

Correct:

```
greet()
```

Mistake 2: Forgetting Parentheses

Incorrect:

```
greet
```

Correct:

```
greet()
```

For a normal function call, use parentheses.

Mistake 3: Forgetting the Colon

Incorrect:

```
def greet()
```

Correct:

```
def greet():
```

Mistake 4: Incorrect Indentation

Incorrect:

```
def greet():
print("Hello!")
```

Correct:

```
def greet():
    print("Hello!")
```

Mistake 5: Calling an Undefined Function

```
calculate_total()
```

If `calculate_total` has not been defined, Python will raise an error.

Mistake 6: Passing the Wrong Number of Arguments

Function:

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

Incorrect:

```
add(10)
```

There are two required parameters but only one argument.

Correct:

```
add(10, 20)
```

---

29. How to Design a Function

Before writing a function, ask three questions:

1. What should this function do?

Example:

```
Calculate the total price.
```

2. What information does it need?

```
price
quantity
```

3. What should it produce?

```
total price
```

Then write:

```
def calculate_total(price, quantity):
    return price * quantity
```

This approach prevents you from creating confusing functions.

---

30. A Function Should Have a Clear Purpose

Compare these:

```
def calculate_total(price, quantity):
    return price * quantity
```

This function has a clear purpose.

Now imagine:

```
def process_everything():
    # calculate price
    # create user
    # send email
    # save data
    # display report
```

This is much harder to understand and maintain.

A good function should generally focus on one clear task.

---

31. Mini Project: Simple Greeting System

Create a small program using a function.

```
def greet_user(name):
    print("Welcome,", name)
    print("Have a great day!")
```

Call it:

```
greet_user("Hafsa")
greet_user("Asim")
```

Output:

```
Welcome, Hafsa
Have a great day!

Welcome, Asim
Have a great day!
```

The function is defined once but reused multiple times.

---

32. 5 Minute Challenge

Create these three functions:

```
show_title()
show_category()
```

```
show_message()
```

Make them display:

```
Python Functions
Programming
Learn once. Reuse everywhere.
```

Then call all three functions.

Expected structure:

```
def show_title():
    print("Python Functions")


def show_category():
    print("Programming")


def show_message():
    print("Learn once. Reuse everywhere.")


show_title()
show_category()
show_message()
```

---

33. Practice Exercises

Exercise 1

Create a function called:

```
say_hello()
```

It should print:

```
Hello, Python!
```

Call it three times.

Exercise 2

Create:

```
show_name(name)
```

It should print the name passed to it.

Exercise 3

Create:

```
multiply(a, b)
```

Return the multiplication result.

Test:

```
print(multiply(6, 7))
```

Expected:

```
42
```

Exercise 4

Create:

```
calculate_area(length, width)
```

Return the area of a rectangle.

Exercise 5

Create:

```
display_student(name, grade)
```

It should display both the student's name and grade.

---

34. Mini Quiz

Question 1

Which keyword is used to define a function?

A. `function`

B. `define`

C. `def`

D. `create`

Question 2

How do you call a function named `greet`?

A.

```
call greet
```

B.

```
greet()
```

C.

```
def greet()
```

D.

```
run greet
```

Question 3

When does the code inside a function normally execute?

A. When the function is defined

B. When Python starts

C. When the function is called

D. Never

Question 4

What does this define?

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

A function with:

A. No parameters

B. One parameter

C. Two parameters

D. Three parameters

Answers

```
1. C
2. B
3. C
4. C
```

---

35. Interview Questions

Beginner

1. What is a function?
2. How do you define a function in Python?
3. How do you call a function?
4. What is the purpose of the `def` keyword?
5. What is the function body?
6. Why is indentation important inside a function?
7. Can a function be called multiple times?
8. What happens if a function is defined but never called?

Intermediate

9. What is the difference between defining and calling a function?
10. What are parameters?
11. What are arguments?
12. What happens when a function is called?
13. Can one function call another function?
14. What happens if the wrong number of arguments is passed?
15. Why should functions have clear responsibilities?

---

36. Cheat Sheet

```
# Define
def greet():
    print("Hello!")


# Call
greet()


# Define with a parameter
def greet(name):
    print("Hello", name)


# Call with an argument
greet("Hafsa")


# Multiple parameters
```

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


# Call and store the returned value
result = add(10, 20)

print(result)
```

Remember:

```
def function_name():
    ↓
Defines the function

function_name()
    ↓
Calls the function
```

With parameters:

```
def greet(name):
       ↑
   parameter


greet("Hafsa")
      ↑
   argument
```

---

37. Key Takeaways

- `def` is used to define a function.
- Defining a function creates its instructions.
- Defining a function does not normally execute its body.
- A function is executed when it is called.
- A function is called using its name followed by `()`.
- Functions can be called multiple times.
- Functions can receive input through parameters.
- Arguments provide actual values to parameters.
- Functions can return values using `return`.
- Function names should clearly describe their purpose.

- Python normally uses `snake_case` for function names.
- Indentation defines the function body.
- Functions help break large programs into smaller, reusable pieces.

The core idea is:

```
DEFINE
   ↓
Create the instructions

CALL
   ↓
Run the instructions

REUSE
   ↓
Call the function whenever needed
```

Visit haas.dev for more resources and guides.

https://dev-roast-app.vercel.app

Next Step

After learning how to define and call functions, the next important topic is:

Function Parameters and Arguments

This will explain how functions receive information and how different types of arguments can be passed to them.