Return Values

Introduction

A function can do more than simply perform an action.

It can also **give a result back** to the code that called it.

This result is called a **return value**.

For example:

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

When we call:

```
result = add(10, 20)
```

the function calculates:

```
10 + 20 = 30
```

and sends `30` back.

So:

```
Arguments
    ↓
Function
    ↓
Processing
    ↓
Return Value
```

Understanding return values is important because they allow functions to work together and allow us to reuse calculated results.

---

1. What Is a Return Value?

A **return value** is the value that a function sends back to the code that called it.

Python uses the `return` keyword.

Example:

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

When we call:

```
add(5, 3)
```

the function returns:

```
8
```

The returned value can then be stored, printed, compared, or used in another calculation.

---

2. Basic `return` Syntax

The basic structure is:

```
def function_name():
    return value
```

Example:

```
def get_number():
    return 10
```

Calling:

```
result = get_number()

print(result)
```

Output:

```
10
```

The function gives `10` back.

---

3. `return` Sends Data Back

Think of `return` like handing something back.

```
You call the function
        ↓
Function does some work
        ↓
Function returns a value
        ↓
You receive the value
```

Example:

```
def get_name():
    return "Hafsa"
```

Then:

```
name = get_name()

print(name)
```

Output:

```
Hafsa
```

The function returned the string `"Hafsa"`.

---

4. `print()` vs `return`

This is one of the most important differences to understand.

Using `print()`

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

Calling:

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

displays:

```
30
```

But the function did not return the value for us to store and use.

Using `return`

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

Now:

```
result = add(10, 20)
```

The value `30` is returned and stored in `result`.

```
print(result)
```

Output:

```
30
```

Remember:

```
print() → displays a value
return  → sends a value back
```

---

5. Why Is `return` Important?

Without `return`, a function can perform an action.

With `return`, a function can produce a result that other parts of the program can use.

For example:

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

Now we can do:

```
total = calculate_total(500, 3)

discount = total * 0.10

final_price = total - discount

print(final_price)
```

Output:

```
1350.0
```

The returned value became input for the next calculation.

---

6. Storing a Return Value

A returned value can be stored in a variable.

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


result = square(6)

print(result)
```

Output:

```
36
```

The flow is:

```
square(6)
    ↓
6 × 6
    ↓
36
    ↓
result
```

---

7. Using a Return Value Directly

You don't always need to store the result.

You can use the function call directly.

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


print(square(6))
```

Output:

```
36
```

The returned value is passed directly to `print()`.

---

8. Using a Return Value in Another Calculation

Returned values can be used in expressions.

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


result = add(10, 20) * 2

print(result)
```

Output:

```
60
```

The process is:

```
add(10, 20)
     ↓
     30
     ↓
30 × 2
     ↓
60
```

---

9. Returning Different Data Types

A function can return many types of values.

String

```
def get_name():
    return "Hafsa"
```

Integer

```
def get_age():
    return 25
```

Float

```
def get_price():
    return 499.99
```

Boolean

```
def is_logged_in():
    return True
```

List

```
def get_skills():
    return ["Python", "JavaScript", "SQL"]
```

The return value depends on what the function needs to provide.

---

10. Returning a Boolean

Functions can return `True` or `False`.

Example:

```
def is_even(number):
    return number % 2 == 0
```

Now:

```
result = is_even(10)

print(result)
```

Output:

```
True
```

We can also use it in a condition:

```
if is_even(10):
    print("Even number")
```

Output:

```
Even number
```

This shows how useful return values can be.

---

11. Returning a List

A function can return an entire list.

```
def get_languages():
    return ["Python", "JavaScript", "Dart"]
```

Then:

```
languages = get_languages()

print(languages)
```

Output:

```
['Python', 'JavaScript', 'Dart']
```

We can also use the list:

```
languages = get_languages()
```

```
for language in languages:
    print(language)
```

Output:

```
Python
JavaScript
Dart
```

---

12. Returning a Dictionary

A function can return a dictionary.

```
def get_user():
    return {
        "name": "Hafsa",
        "role": "Developer"
    }
```

Then:

```
user = get_user()

print(user["name"])
```

Output:

```
Hafsa
```

This is common in real applications when functions need to provide structured information.

---

13. Returning Multiple Values

Python allows a function to return multiple values.

Example:

```
def calculate(a, b):
    return a + b, a - b
```

Call:

```
addition, subtraction = calculate(10, 5)

print(addition)
print(subtraction)
```

Output:

```
15
5
```

Python actually packages these returned values into a tuple.

Conceptually:

```
return (15, 5)
```

Then they are unpacked into:

```
addition = 15
subtraction = 5
```

---

14. `return` Immediately Stops a Function

When Python reaches a `return` statement, the function stops executing.

Example:

```
def test():
    print("Start")
    return 10
    print("End")
```

Call:

```
result = test()

print(result)
```

Output:

```
Start
10
```

`"End"` is never printed.

Why?

Because the function already returned.

Think:

```
Function starts
     ↓
Code runs
```

```
     ↓
return
     ↓
Function ends
```

---

15. Multiple Return Statements

A function can contain multiple `return` statements.

Usually, they are used with conditions.

Example:

```
def check_number(number):
    if number > 0:
        return "Positive"

    if number < 0:
        return "Negative"

    return "Zero"
```

Now:

```
print(check_number(10))
print(check_number(-5))
print(check_number(0))
```

Output:

```
Positive
Negative
Zero
```

Only one return path is reached for each function call.

---

16. `return` Inside an `if`

This is very common.

```
def get_status(age):
    if age >= 18:
        return "Adult"
    else:
        return "Minor"
```

Call:

```
print(get_status(25))
```

Output:

```
Adult
```

Another call:

```
print(get_status(15))
```

Output:

```
Minor
```

---

17. Returning the Result of a Calculation

You don't always need a separate variable.

Instead of:

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

you can write:

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

Both work.

The second version is shorter because the calculated expression can be returned directly.

---

18. Returning a Variable

You can also return a variable.

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

This is useful when the function performs several steps.

Example:

```
def calculate_final_price(price, discount):
    discount_amount = price * discount / 100
```

```
    final_price = price - discount_amount

    return final_price
```

Call:

```
print(calculate_final_price(1000, 10))
```

Output:

```
900.0
```

---

19. What Happens If There Is No `return`?

Consider:

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

This function does not explicitly return a value.

Python automatically returns:

```
None
```

Example:

```
result = greet()

print(result)
```

Output:

```
Hello
None
```

`None` means that the function has no meaningful return value.

---

20. `return` Without a Value

You can write:

```
def check_age(age):
    if age < 0:
        return

    print("Valid age")
```

Here:

```
return
```

stops the function without returning a useful value.

The return value is:

```
None
```

This can be useful when you simply want to exit a function early.

---

21. Returning `None`

You can explicitly return `None`.

```
def get_data():
    return None
```

This means:

> There is currently no value to return.

You may encounter `None` frequently when working with real Python programs.

---

22. Function Composition

One of the most powerful uses of return values is allowing functions to work together.

Example:

```
def get_price():
    return 1000


def calculate_discount(price):
    return price * 0.10


price = get_price()

discount = calculate_discount(price)

print(discount)
```

Output:

```
100.0
```

One function produces a value.

Another function receives that value.

```
get_price()
     ↓
   1000
     ↓
calculate_discount()
     ↓
   100
```

This is called **function composition**: using the result of one function as input to another.

---

23. Functions Can Be Chained

We can also use one function's return value directly inside another function call.

```
def double(number):
    return number * 2


def square(number):
    return number * number


result = square(double(3))

print(result)
```

Let's follow the flow:

```
double(3)
   ↓
6
   ↓
square(6)
   ↓
36
```

Output:

```
36
```

This is possible because `double()` returns a value.

---

24. Practical Example: Student Result

Let's create a function that calculates a student's average.

```
def calculate_average(math, english, computer):
    total = math + english + computer
    average = total / 3

    return average
```

Now:

```
average = calculate_average(80, 90, 85)

print("Average:", average)
```

Output:

```
Average: 85.0
```

The function does not decide how the result should be displayed.

It simply calculates and returns the value.

This separation makes the function reusable.

---

25. Practical Example: Grade Calculator

We can use a return value to determine a grade.

```
def calculate_grade(average):
    if average >= 90:
        return "A"
    elif average >= 80:
        return "B"
    elif average >= 70:
        return "C"
    else:
        return "Needs Improvement"
```

Now:

```
average = 85

grade = calculate_grade(average)

print(grade)
```

Output:

```
B
```

Here one piece of information flows through the program:

```
Average
   ↓
calculate_grade()
   ↓
Grade
```

---

26. Practical Example: haas.dev Resource Data

Imagine a function that creates information about a resource.

```
def get_resource():
    return {
        "title": "Python Functions",
        "category": "Python",
        "level": "Beginner"
    }
```

Then:

```
resource = get_resource()

print(resource["title"])
print(resource["category"])
```

Output:

```
Python Functions
Python
```

The function returns structured data that another part of the application can use.

This pattern is common in web applications and APIs.

---

27. `print()` and `return` Together

Sometimes a function can both print and return.

```
def add(a, b):
    result = a + b
```

```
    print("Calculating...")

    return result
```

Call:

```
answer = add(10, 20)

print(answer)
```

Output:

```
Calculating...
30
```

The function displays a message and also returns the actual result.

However, in reusable functions, it is often better to keep calculation logic separate from display logic.

---

28. A Better Design

Instead of:

```
def calculate_total(price, quantity):
    total = price * quantity
    print("Total:", total)
```

prefer when the result needs to be reused:

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

Then the caller decides what to do:

```
total = calculate_total(500, 3)

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

Or:

```
total = calculate_total(500, 3)

if total > 1000:
    print("Large order")
```

The function becomes more flexible.

---

29. Common Mistakes

Mistake 1: Expecting `print()` to Return a Value

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

Then:

```
result = add(10, 20)

print(result)
```

Output:

```
30
None
```

Why?

Because `print()` displayed `30`, but the function itself did not return `30`.

Mistake 2: Forgetting `return`

Incorrect:

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

Correct:

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

Mistake 3: Code After `return`

```
def test():
    return 10
    print("Hello")
```

The `print()` will never execute.

Mistake 4: Returning Too Early

Consider:

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

The second return can never be reached.

Once Python executes the first `return`, the function ends.

Mistake 5: Forgetting to Store the Result

This is valid:

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

But if you need the result later, store it:

```
result = add(10, 20)
```

---

30. Problem Solving Framework

When creating a function, think:

Step 1: What does the function receive?

```
Inputs
```

Step 2: What does it do?

```
Processing
```

Step 3: What result should it produce?

```
Output
```

Example:

```
Input:
price, quantity

Processing:
price × quantity

Output:
total
```

Code:

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

---

31. The Function Pipeline

A useful mental model is:

```
INPUT
  ↓
PARAMETERS
  ↓
PROCESSING
  ↓
RETURN
  ↓
OUTPUT
```

Example:

```
def calculate_area(length, width):
    return length * width
```

Flow:

```
10, 5
 ↓
length = 10
width = 5
 ↓
10 × 5
 ↓
return 50
 ↓
area = 50
```

---

32. Mini Project: Shopping Cart Calculator

Create a small shopping cart calculation system.

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


def calculate_discount(subtotal, discount):
```

```
    return subtotal * discount / 100


def calculate_final_price(subtotal, discount_amount):
    return subtotal - discount_amount
```

Use the functions together:

```
subtotal = calculate_subtotal(500, 3)

discount_amount = calculate_discount(subtotal, 10)

final_price = calculate_final_price(
    subtotal,
    discount_amount
)

print("Subtotal:", subtotal)
print("Discount:", discount_amount)
print("Final Price:", final_price)
```

Output:

```
Subtotal: 1500
Discount: 150.0
Final Price: 1350.0
```

Notice how the return value of one function becomes input for another.

```
calculate_subtotal()
        ↓
    subtotal
        ↓
calculate_discount()
        ↓
discount_amount
        ↓
calculate_final_price()
        ↓
    final_price
```

This is how functions can be combined to build larger programs.

---

33. 5 Minute Challenge

Create these functions:

```
calculate_total(price, quantity)
calculate_discount(total, discount)
calculate_final_price(total, discount_amount)
```

Each function should **return** its result.

Then connect them:

```
price + quantity
        ↓
    total
        ↓
discount percentage
        ↓
discount amount
        ↓
final price
```

Finally print:

```
Total:
Discount:
Final Price:
```

The important requirement is:

Do not print the result inside the calculation functions. Return it instead.

---

34. Practice Exercises

Exercise 1

Create:

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

Return the sum.

Exercise 2

Create:

```
def subtract(a, b):
```

Return the difference.

Exercise 3

Create:

```
def multiply(a, b):
```

Return the product.

Exercise 4

Create:

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

Return the division result.

Exercise 5

Create:

```
def is_even(number):
```

Return `True` if the number is even and `False` otherwise.

Exercise 6

Create:

```
def get_full_name(first_name, last_name):
```

Return the complete name.

Exercise 7

Create:

```
def calculate_average(a, b, c):
```

Return the average of three numbers.

Exercise 8

Create:

```
def get_user():
```

Return a dictionary containing:

```
name
age
role
```

---

35. Mini Quiz

Question 1

Which keyword sends a value back from a function?

A. `send`

B. `return`

C. `print`

D. `output`

Question 2

What does this function return?

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

A. Nothing

B. The value of `a` only

C. The sum of `a` and `b`

D. A string

Question 3

What happens after Python executes `return`?

A. The function continues normally

B. The function stops

C. The program always crashes

D. Python repeats the function

Question 4

What does a function return if it reaches the end without a `return` statement?

A. `0`

B. `False`

C. `None`

D. An empty string

Answers

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

---

36. Interview Questions

Beginner

1. What is a return value?
2. Which keyword is used to return a value?
3. What is the difference between `print()` and `return`?
4. Can a function return a string?
5. Can a function return a list?
6. Can a function return a dictionary?
7. What happens when Python reaches a `return` statement?
8. What does a function return if there is no explicit `return`?

Intermediate

9. Can a function have multiple `return` statements?
10. Can a function return multiple values?
11. How can the return value of one function be used by another function?
12. Why is returning a result often more reusable than printing it?
13. What is function composition?
14. What happens to code written after an executed `return` statement?

---

37. Cheat Sheet

Return a calculation

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

Store the result

```
result = add(10, 20)
```

Print the result

```
print(result)
```

Use directly

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

Return a string

```
def get_name():
    return "Hafsa"
```

Return a boolean

```
def is_even(number):
    return number % 2 == 0
```

Return a list

```
def get_skills():
    return ["Python", "JavaScript"]
```

Return a dictionary

```
def get_user():
    return {
        "name": "Hafsa",
        "role": "Developer"
    }
```

Multiple return values

```
def calculate(a, b):
    return a + b, a - b
```

No return

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

This implicitly returns:

```
None
```

---

38. Key Takeaways

- A return value is the result a function sends back.
- Python uses the `return` keyword.

- `return` allows the result of a function to be reused.
- Returned values can be stored in variables.
- Returned values can be passed to other functions.
- Returned values can be used in calculations and conditions.
- Functions can return strings, numbers, booleans, lists, dictionaries, and other objects.
- A function can return multiple values.
- `return` immediately stops the function.
- Code after an executed `return` does not run.
- A function without an explicit return value returns `None`.
- `print()` displays a value; `return` sends a value back.
- Returning values generally makes calculation functions more reusable.

The core idea:

```
              ARGUMENTS
                  ↓
              FUNCTION
                  ↓
              PROCESSING
                  ↓
               return
                  ↓
           RETURN VALUE
                  ↓
        Store / Print / Reuse
```

Visit haas.dev for more resources and guides.

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

Next Step

The next topic is **Default Arguments**, where we will learn how to give function parameters automatic values when the caller does not provide them.