Chapter 6 Functions

Opening Problem

Find the sum of the odd numbers integers:

from 1 to 10

from 20 to 37

from 35 to 49

Problem

```
sum = 0
for i in range(1, 11):
    if i%2 == 1:
        sum = sum + i
print("Sum from 1 to 10 is", sum)

sum = 0
for i in range(20, 38):
    if i%2 == 1:
        sum = sum + i
print("Sum from 20 to 37 is", sum)

sum = 0
for i in range(35, 50):
    if i%2 == 1:
        sum = sum + i
print("Sum from 35 to 49 is", sum)
```

Solution

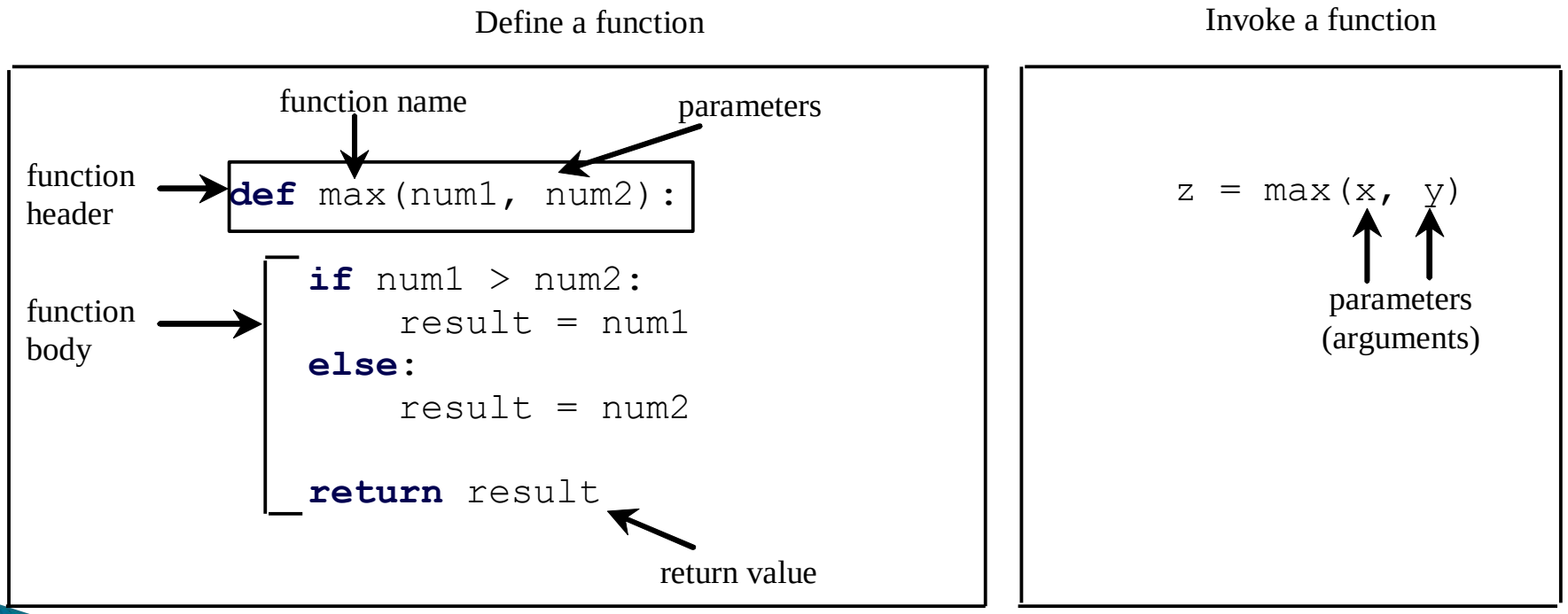
```
def sum(i1, i2):
    result = 0
    for i in range(i1, i2):
        if i%2 == 1:
            result += i
    return result

def main():
    print("Sum from 1 to 10 is", sum(1, 11))
    print("Sum from 20 to 37 is", sum(20, 38))
    print("Sum from 35 to 49 is", sum(35, 50))

main() # Call the main function
```

Objectives

- To define functions
- To invoke value-returning functions
- To invoke functions that do not return a value
- To pass arguments to functions
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain
- To determine the scope of variables
- To return multiple values from a function

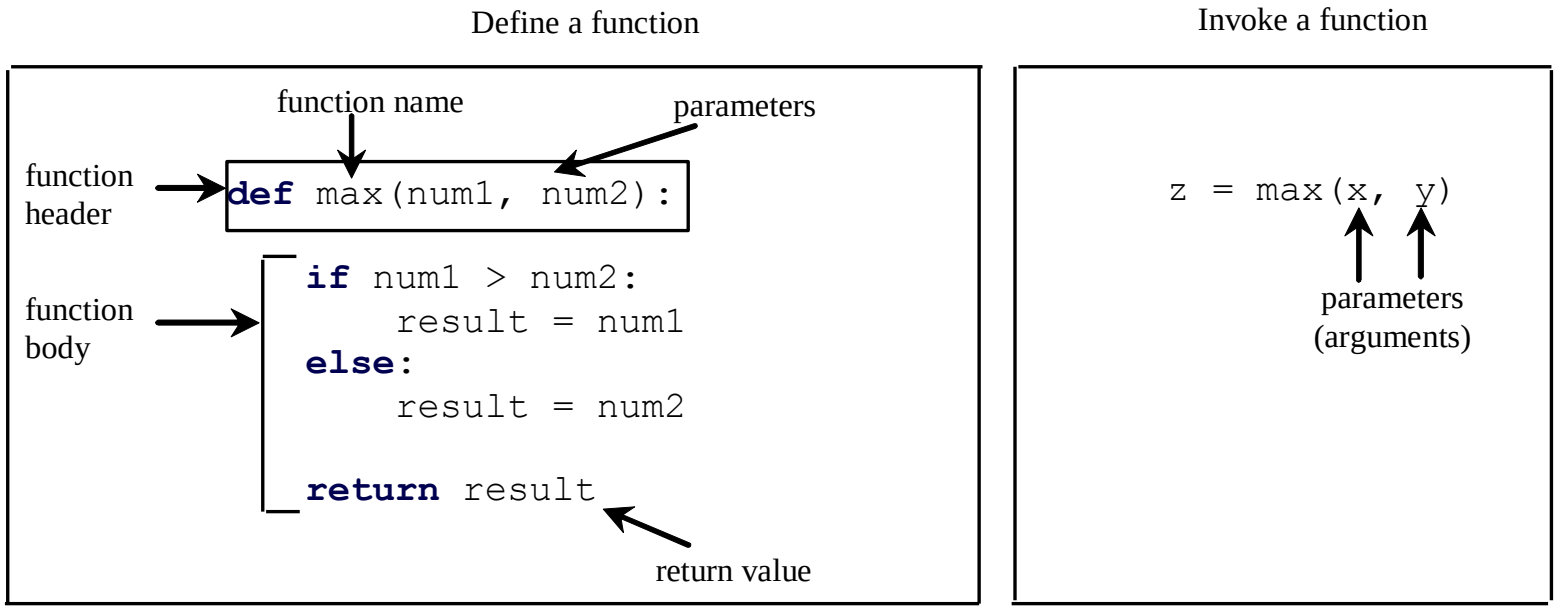
Defining Functions

A function is a collection of statements that are grouped together to perform an operation.



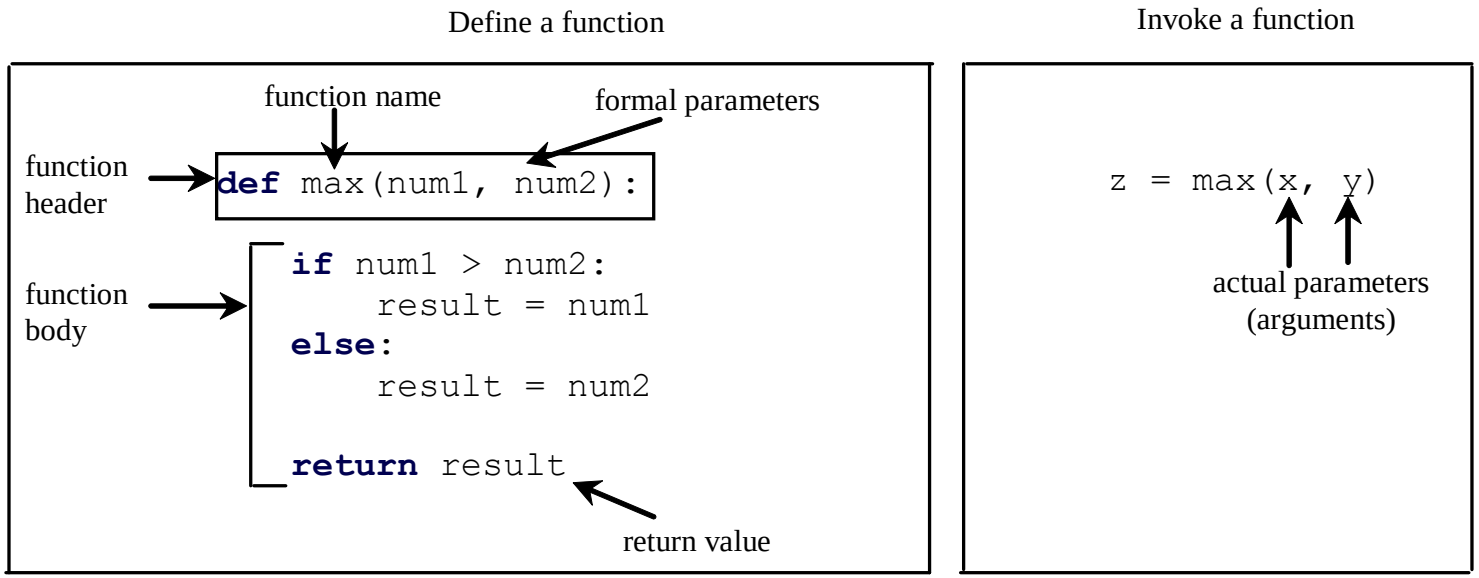
Function Header

A function contains a header and body. The header begins with the **def** keyword, followed by function's name and parameters, followed by a colon.



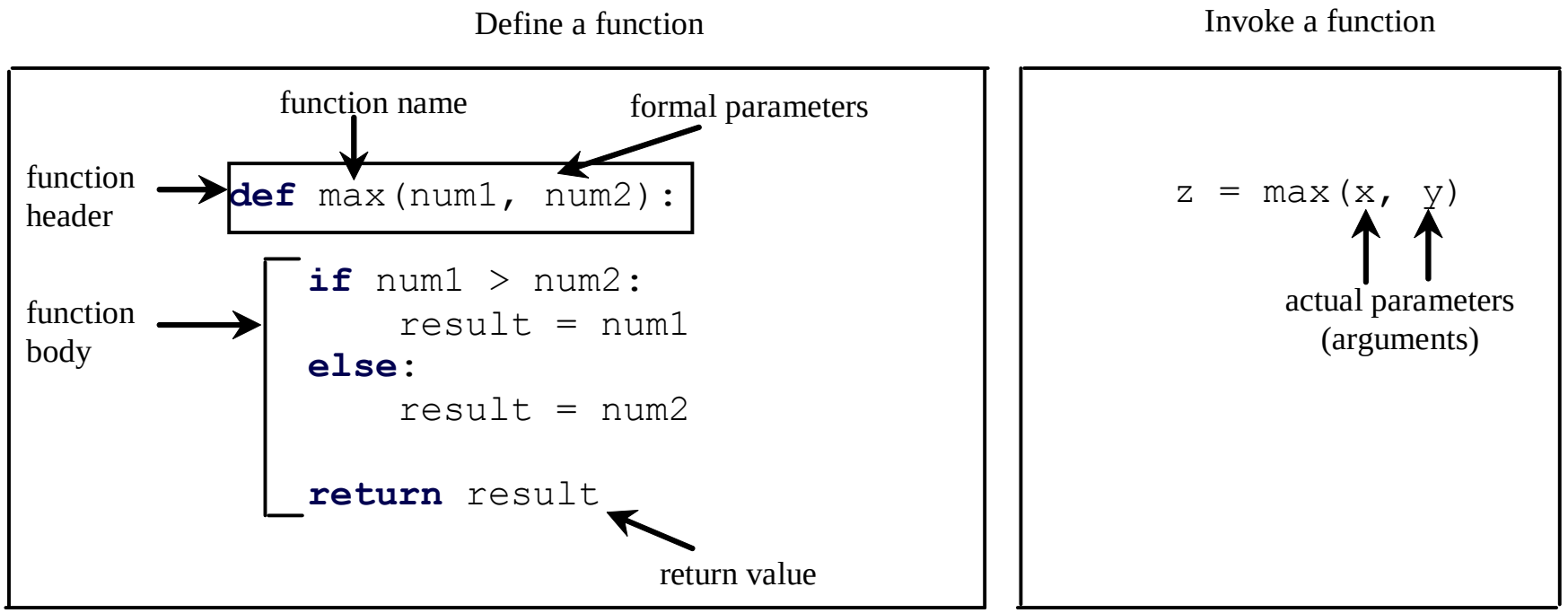
Formal Parameters

The variables defined in the function header are known as *formal parameters*.



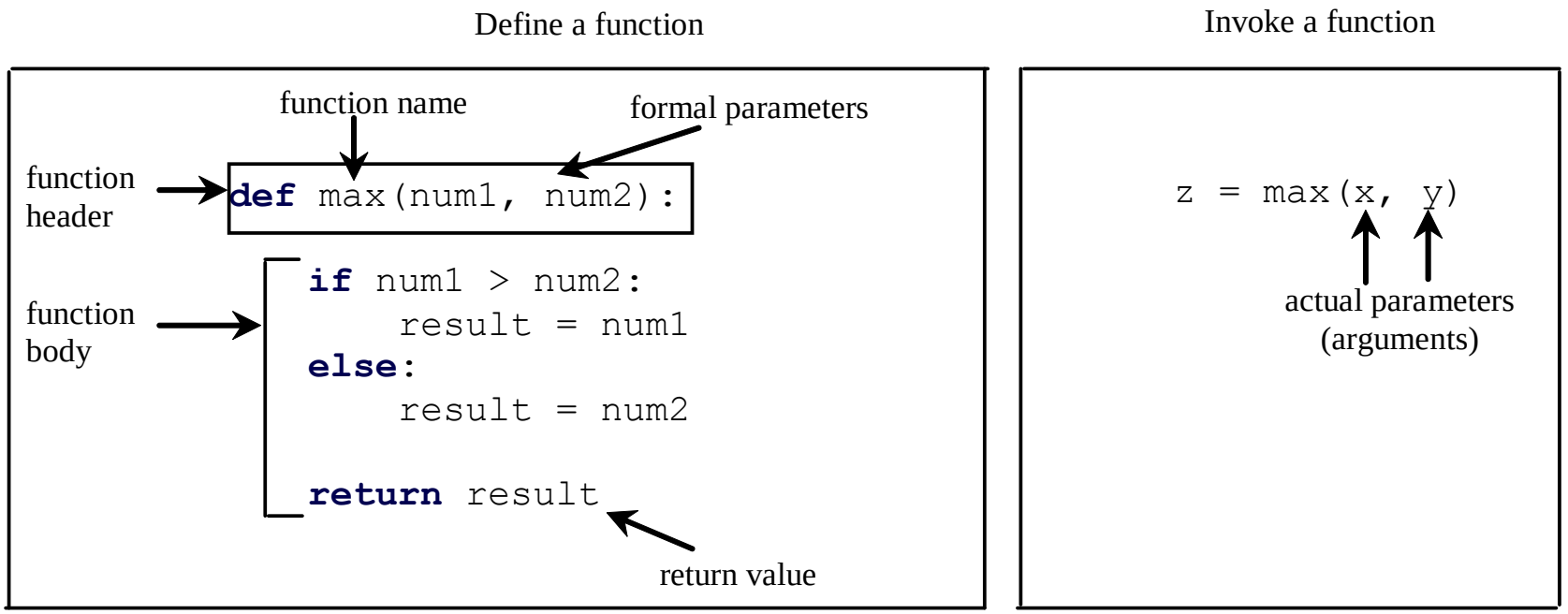
Actual Parameters

When a function is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.



Return Value

A function may return a value using the return keyword.



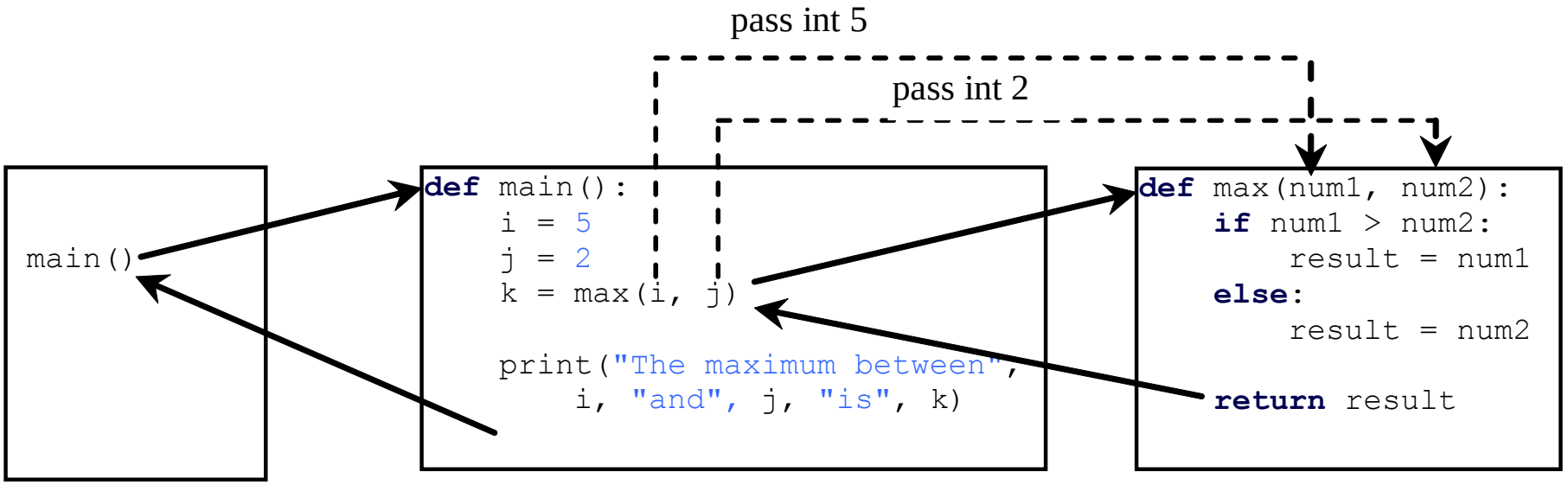
Calling Functions

Testing the **`max`** function (user-defined, not built-in)

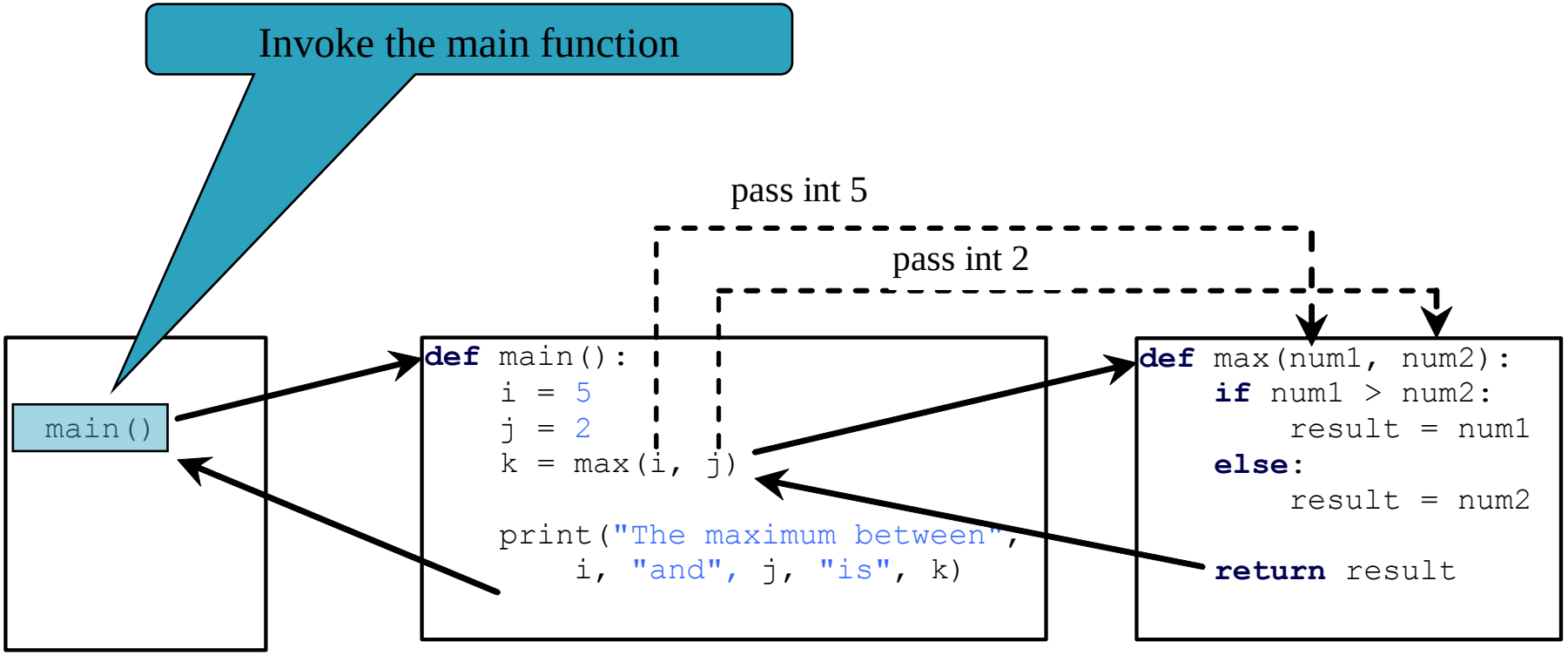
This program demonstrates calling a function **`max`** to return the largest of the **`int`** values

TestMax

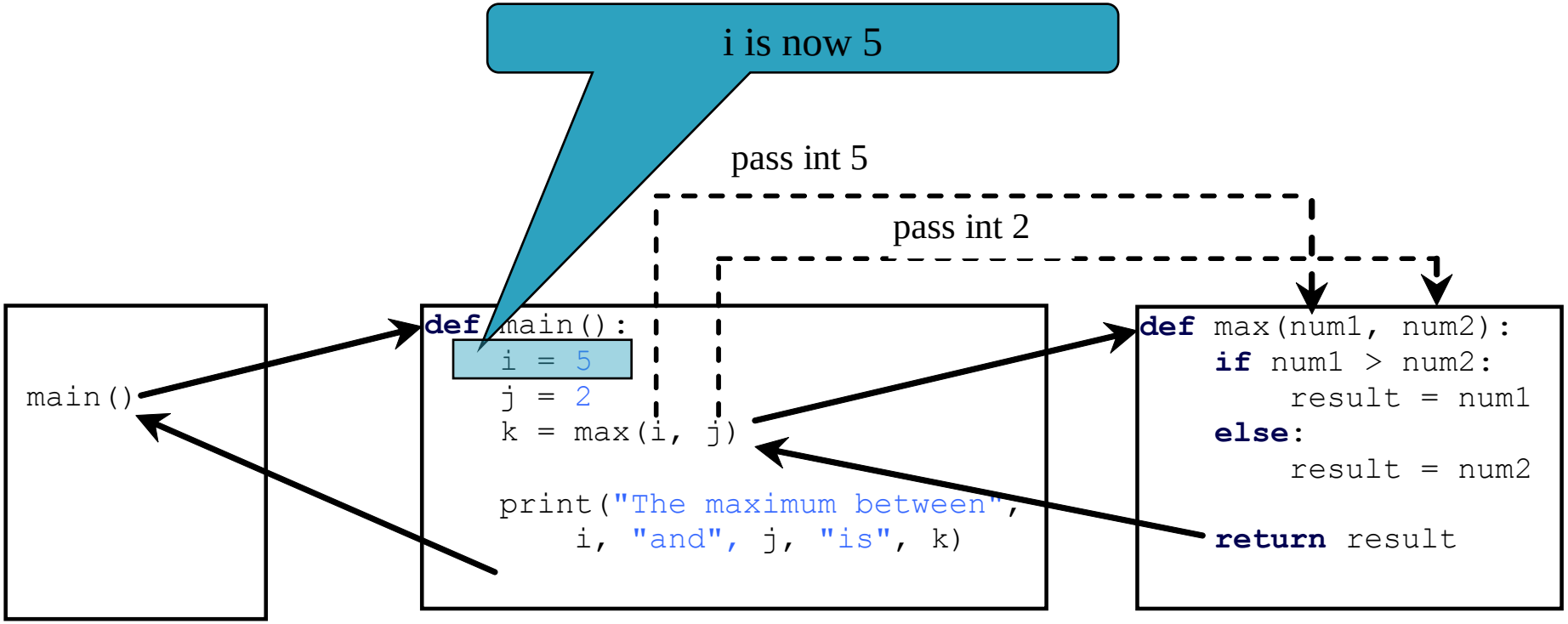
Calling Functions, cont.



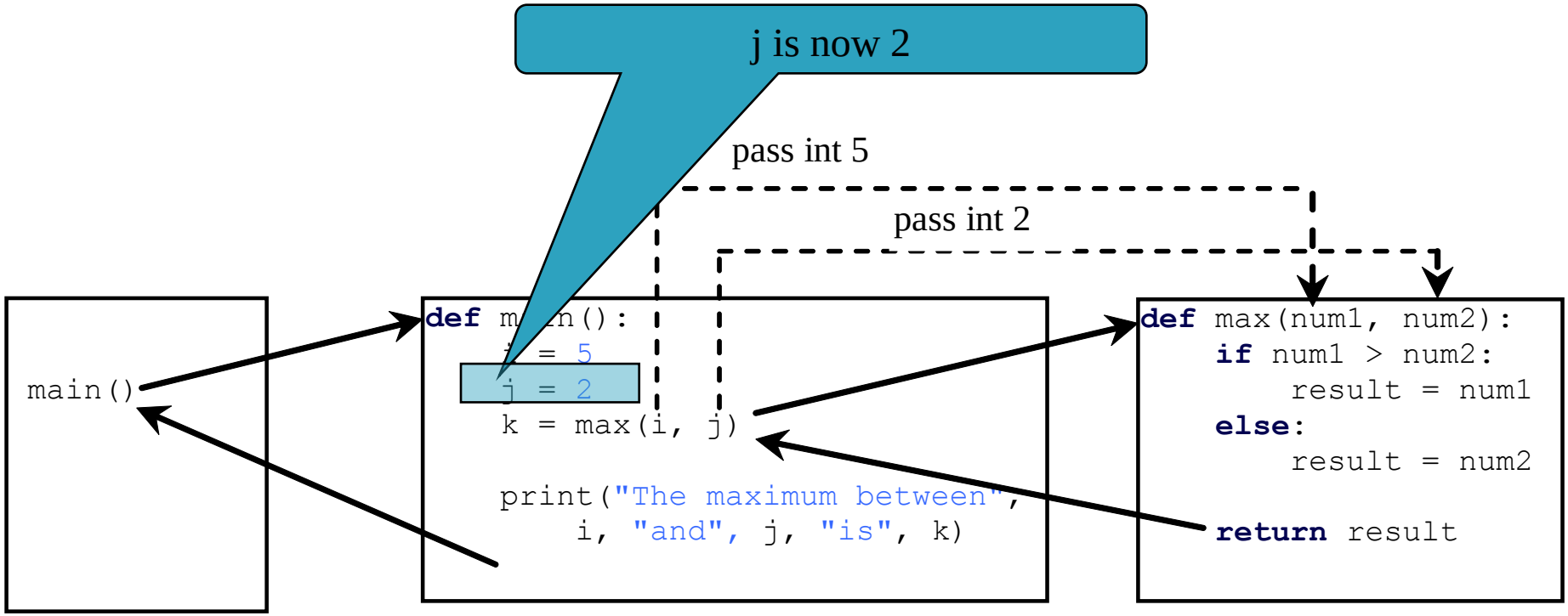
Trace Function Invocation



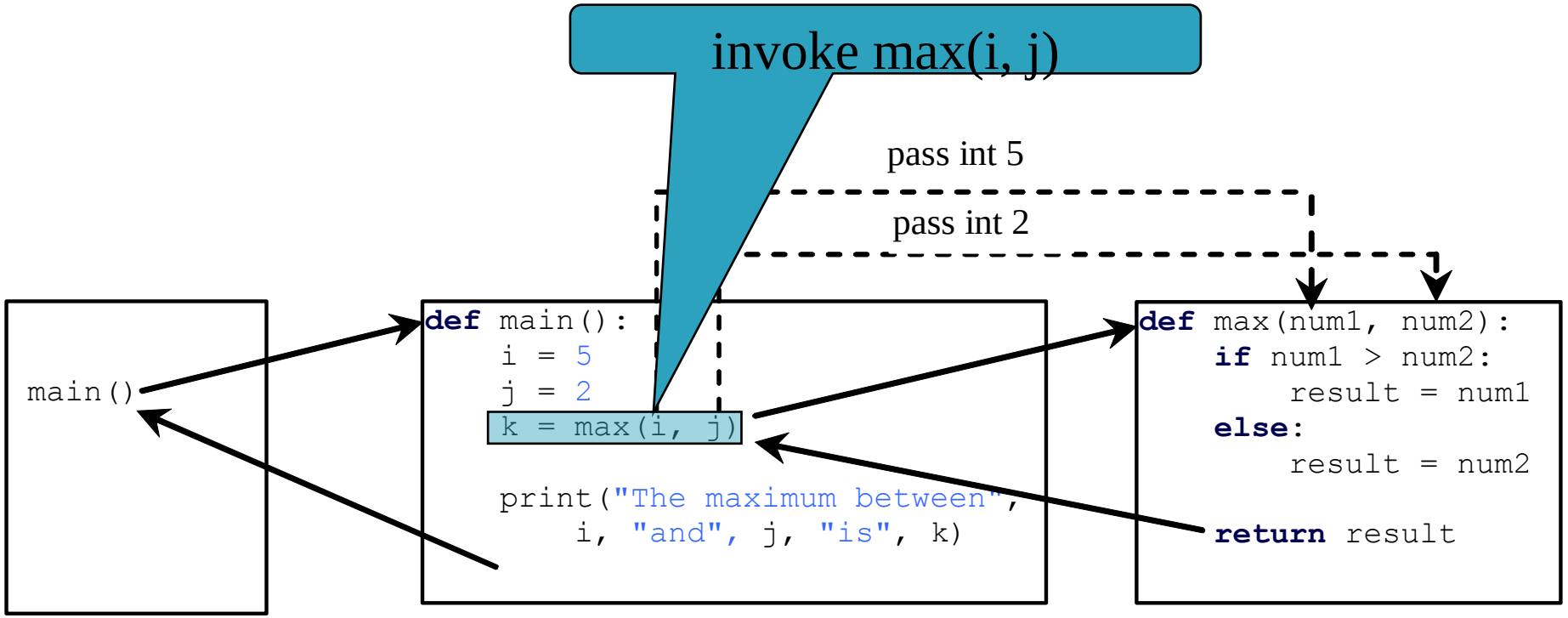
Trace Function Invocation



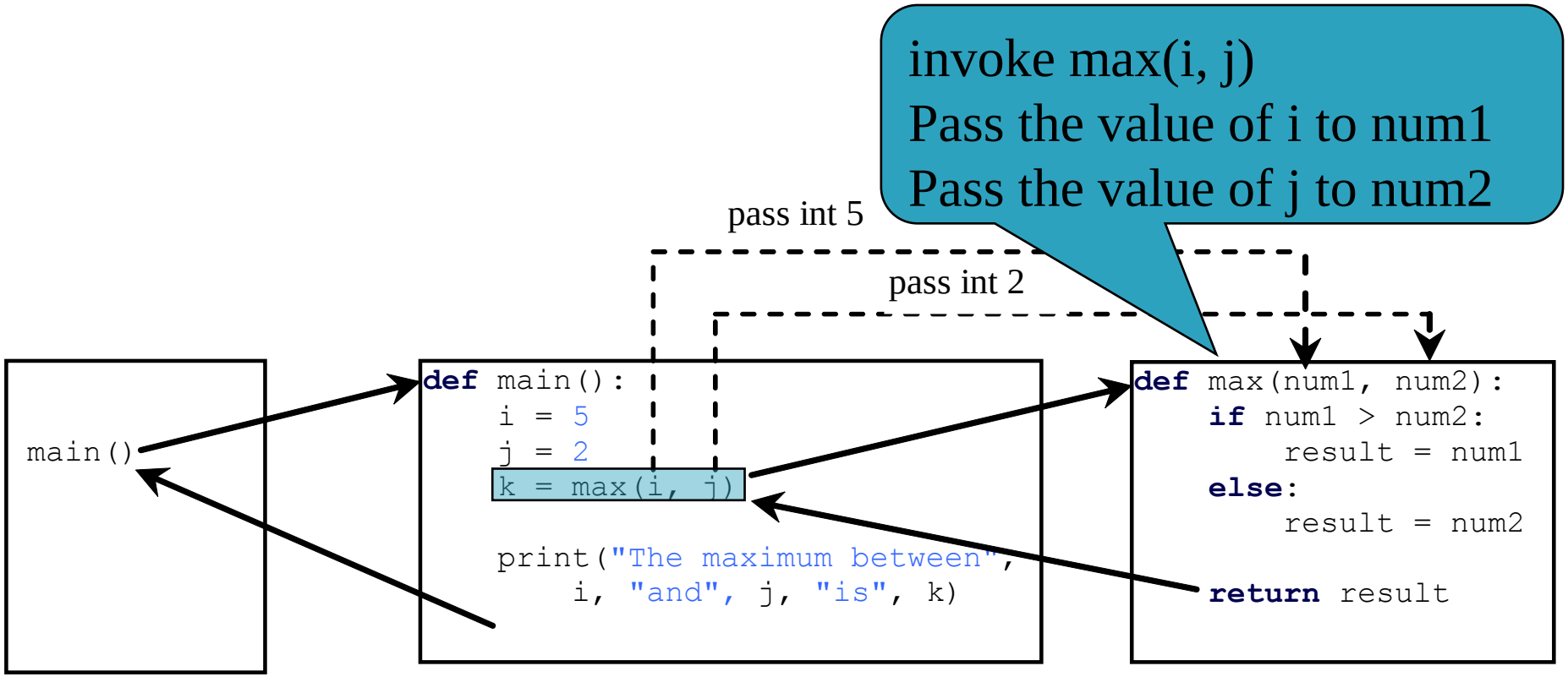
Trace Function Invocation



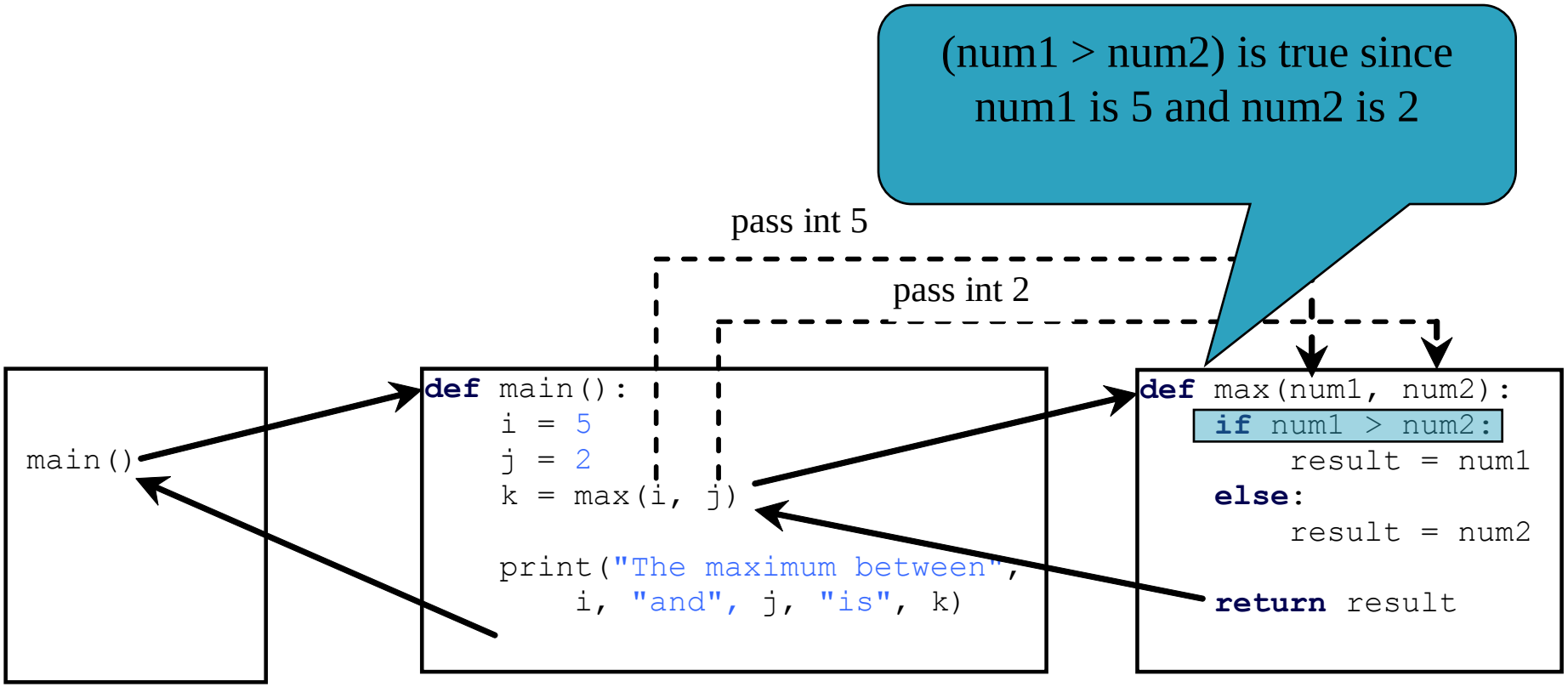
Trace Function Invocation



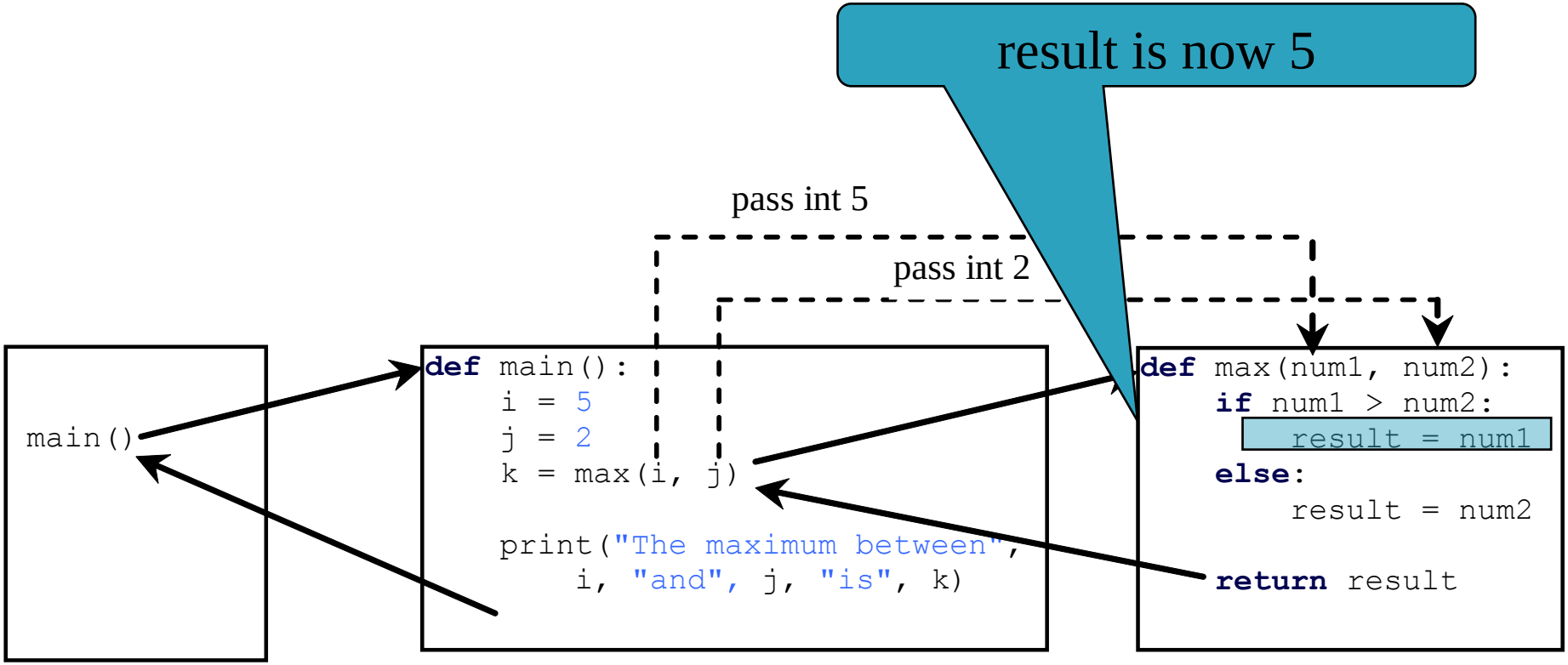
Trace Function Invocation



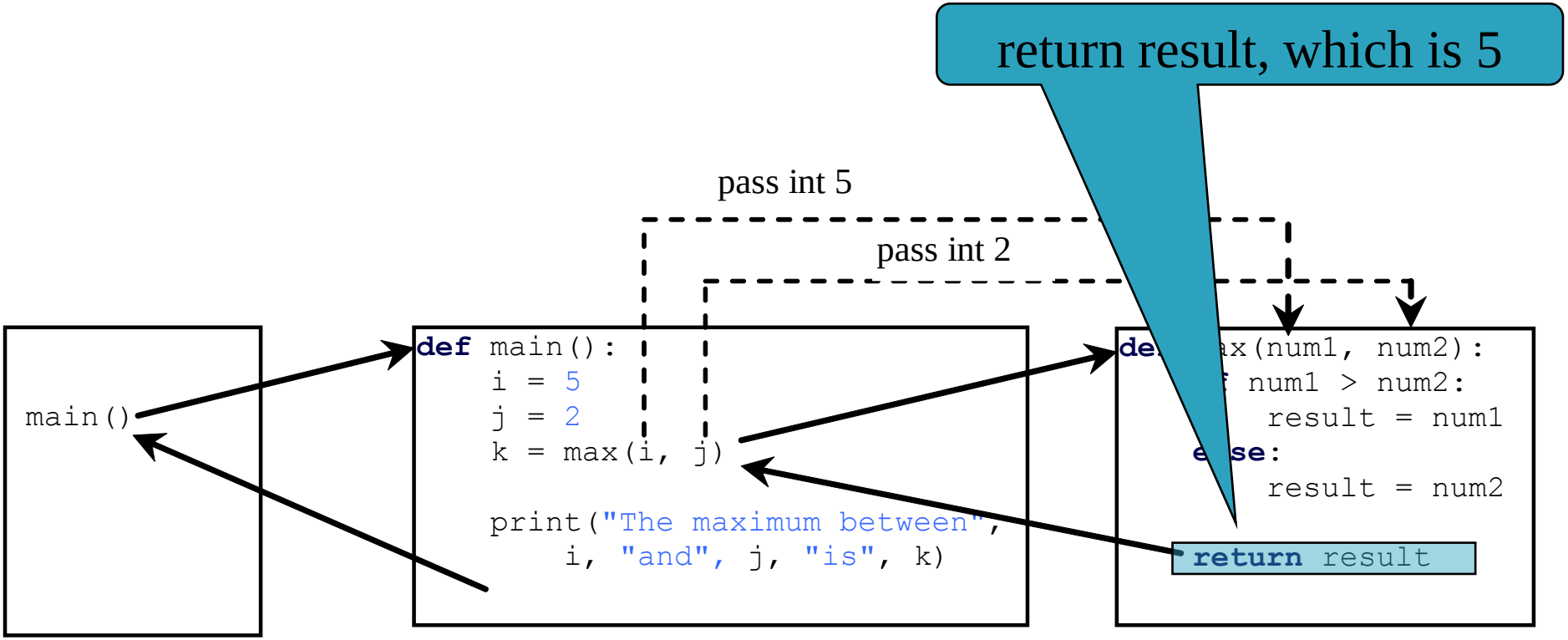
Trace Function Invocation



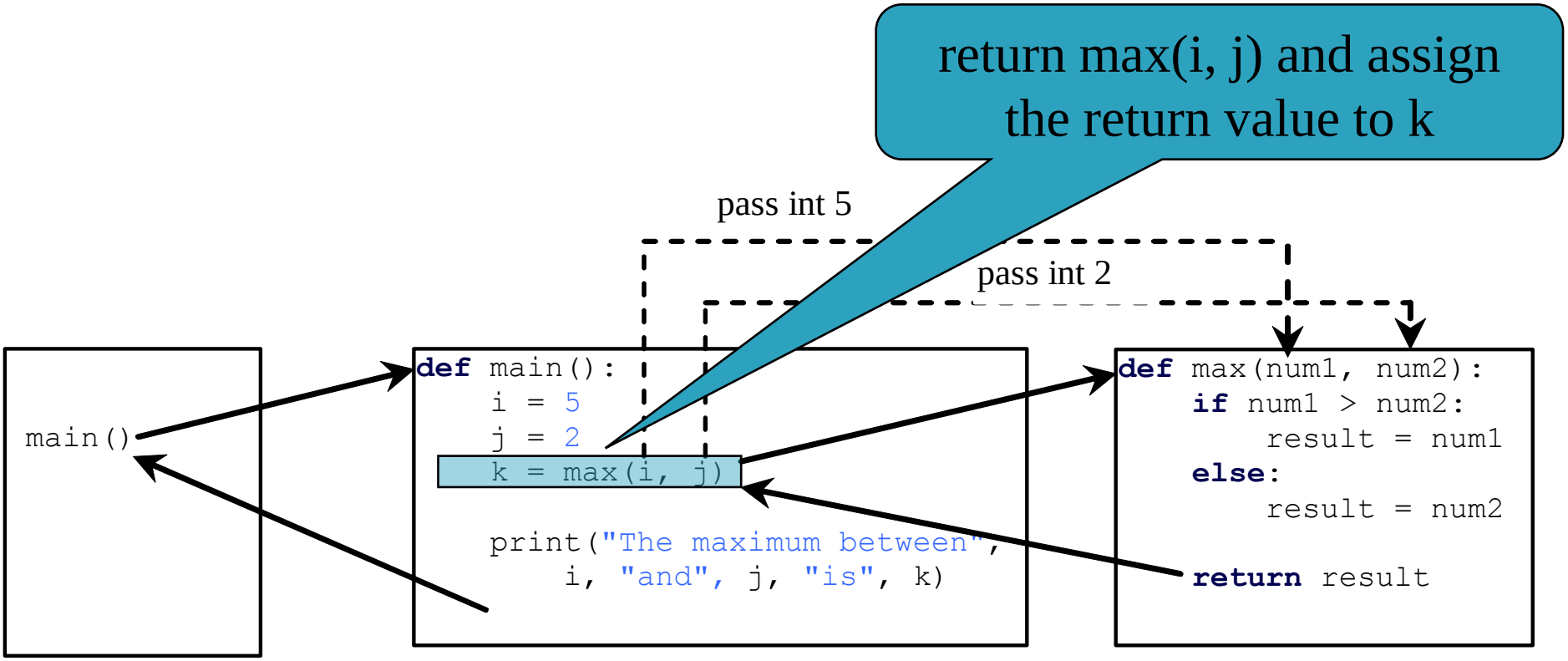
Trace Function Invocation



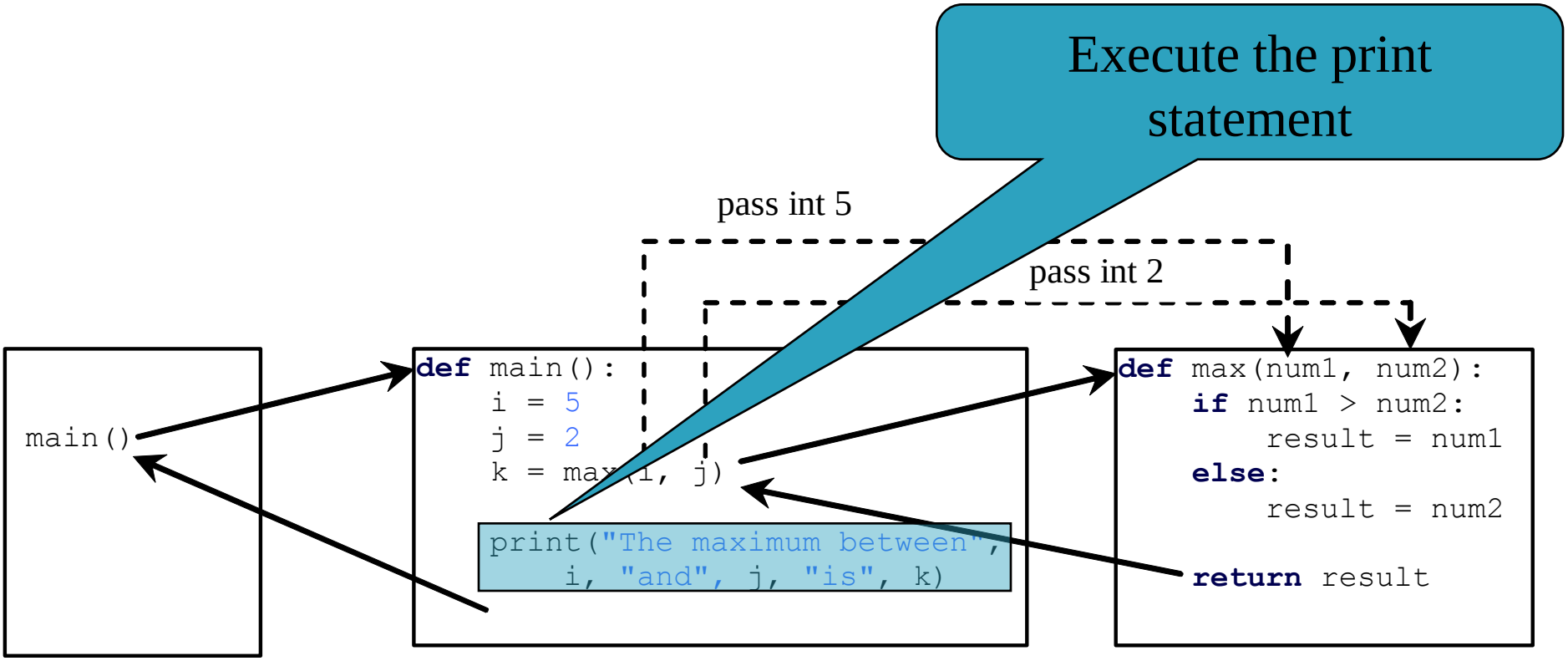
Trace Function Invocation



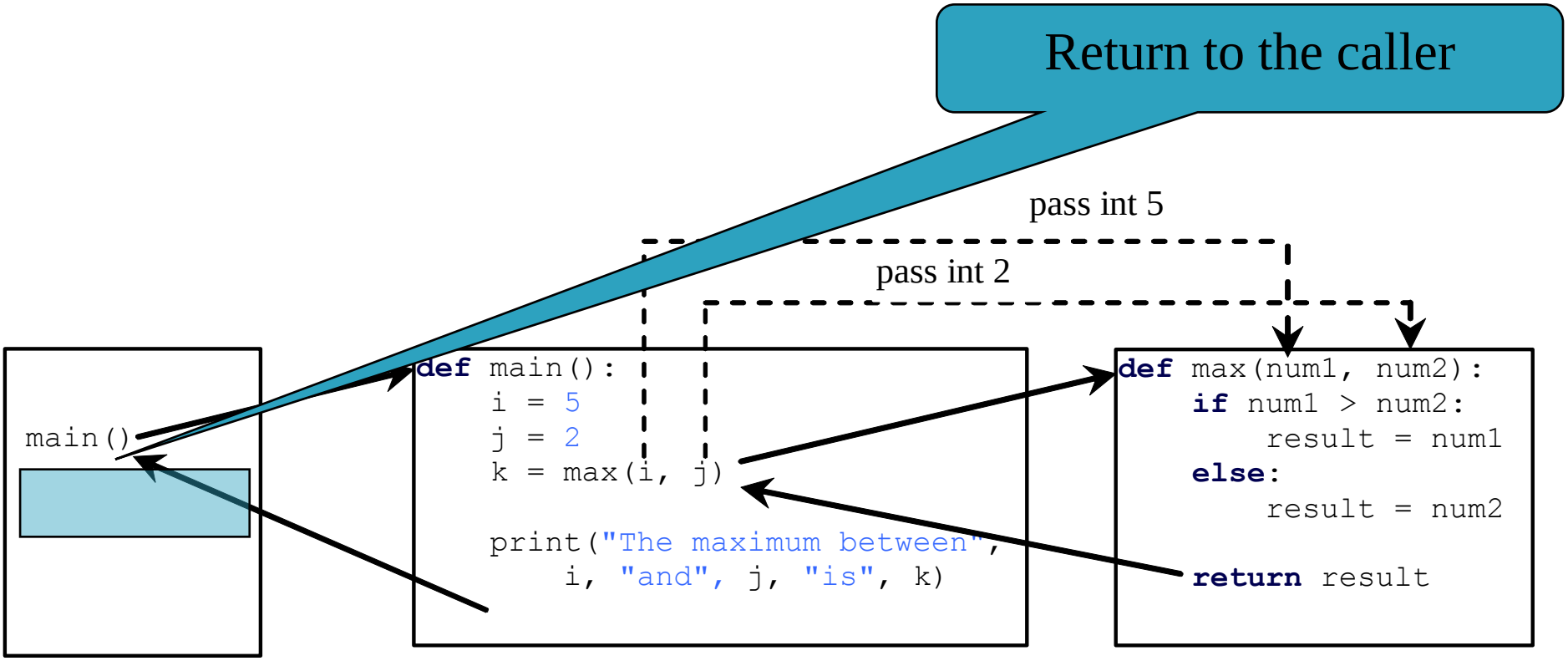
Trace Function Invocation



Trace Function Invocation



Trace Function Invocation



Functions With/Without Return Values

This type of function does not return a value. The function performs some actions.

PrintGradeFunction

ReturnGradeFunction

The None Value

A function that does not return a value is known as a *void* function in other programming languages such as C++ and C#. In Python, such function returns a special None.

```
def sum(number1, number2):
    total = number1 + number2
print(sum(1, 2))
```

Pass by Value

When you invoke a function with a parameter, the value of the argument is passed to the parameter.

This is referred to as *pass-by-value*.

If the argument is a number or a string, the argument is not affected, regardless of the changes made to the parameter inside the function.

Increment

Modularizing Code

Functions can be used to reduce redundant coding and enable code reuse. Functions can also be used to modularize code and improve the quality of the program.

GCDFunction

TestGCDFunction

PrimeNumberFunction

Scope of Variables

Scope: the part of the program where the variable can be referenced.

A variable created inside a function is referred to as a *local variable*. Local variables can only be accessed inside a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.

In Python, you can also use *global variables*. They are created outside all functions and are accessible to all functions in their scope.

Example 1

```
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)
f1()
print(globalVar)
print(localVar)  # Out of scope, Error
```

Example 2

```
x = 1
def f1():
    x = 2
    print(x) # Displays 2, the local x
f1()
print(x) # Displays 1, the global x
```

Example 3

```
sum = 0
for i in range(0, 5):
    sum += i
print(i)
```

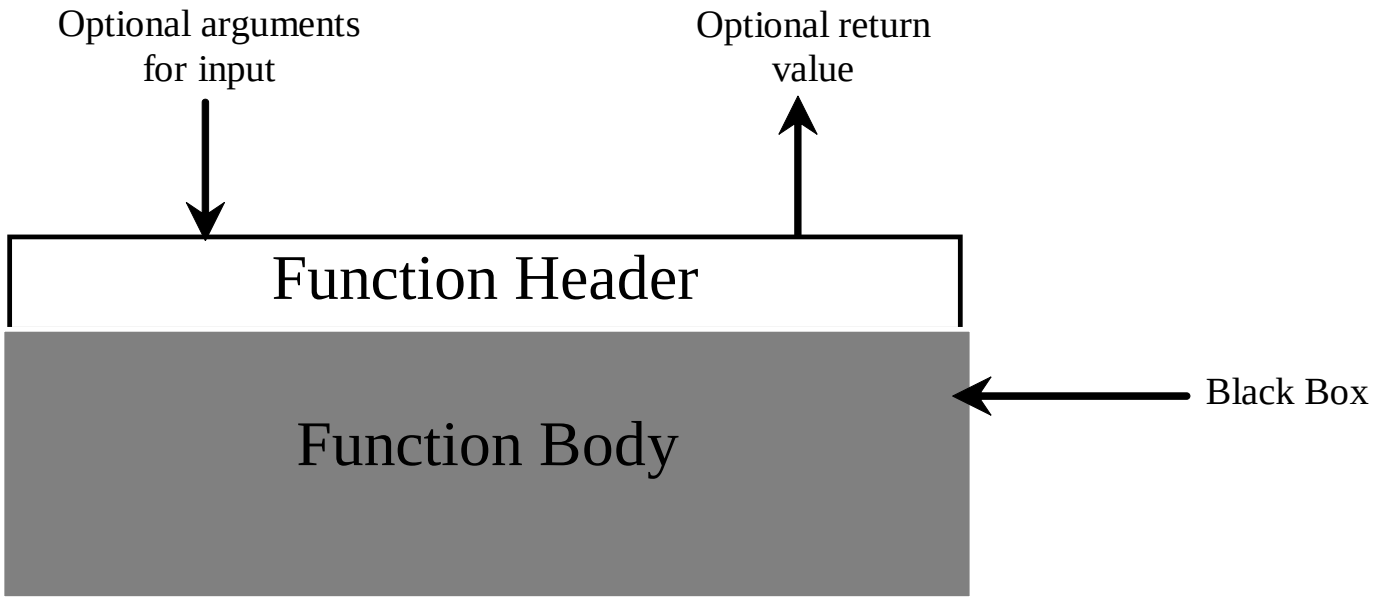
Returning Multiple Values

Python allows a function to return multiple values. Listing 5.9 defines a function that takes two numbers and returns them in non-descending order.

MultipleReturnValueDemo

Function Abstraction

You can think of the function body as a black box that contains the detailed implementation for the function.



Benefits of Functions

- Write a function once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

PrintCalender Case Study

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

PrintCalendar

Problem: Converting Decimals to Hexadecimals

Write a function that converts a decimal integer to a hexadecimal.

Decimal2HexConversion