Chapter 6 Functions

Opening Problem

Find the sum of integers from 1 to 10, from 20 to 37, and from 35 to 49, respectively.

Problem

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

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

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

Problem

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

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

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

Solution

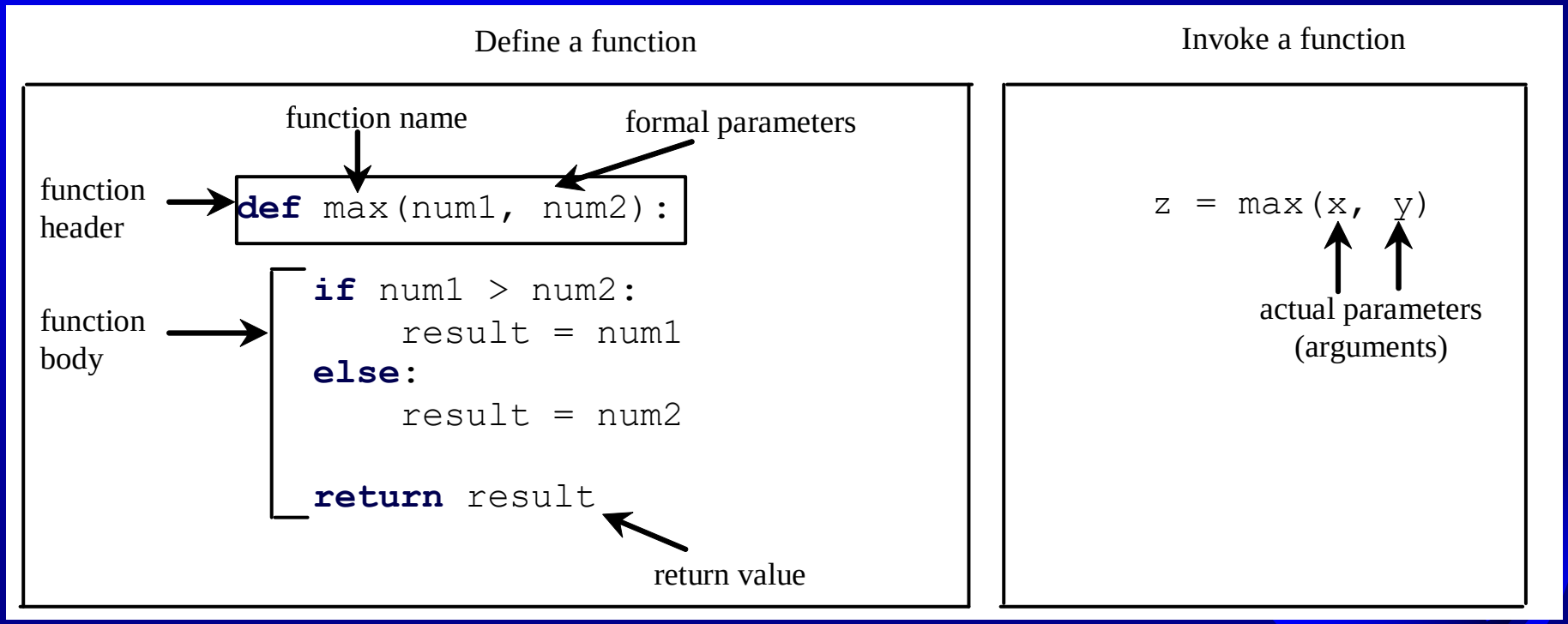
```
def sum(i1, i2):
    result = 0
    for i in range(i1, i2):
        result += i
    return result

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

main() # Call the main function
```

Objectives

- To define functions (§6.2).
- To invoke value-returning functions (§6.3).
- To invoke functions that does not return a value (§6.4).
- To pass arguments by values (§6.5).
- To pass arguments by values (§6.6).
- To develop reusable code that is modular, easy to read, easy to debug, and easy to maintain (§6.7).
- To create modules for reusing functions (§§6.7-6.8).
- To determine the scope of variables (§6.9).
- To define functions with default arguments (§6.10).
- To return multiple values from a function (§6.11).
- To apply the concept of function abstraction in software development (§6.12).
- To design and implement functions using stepwise refinement (§6.13).
- To simplify drawing programs using functions (§6.14).

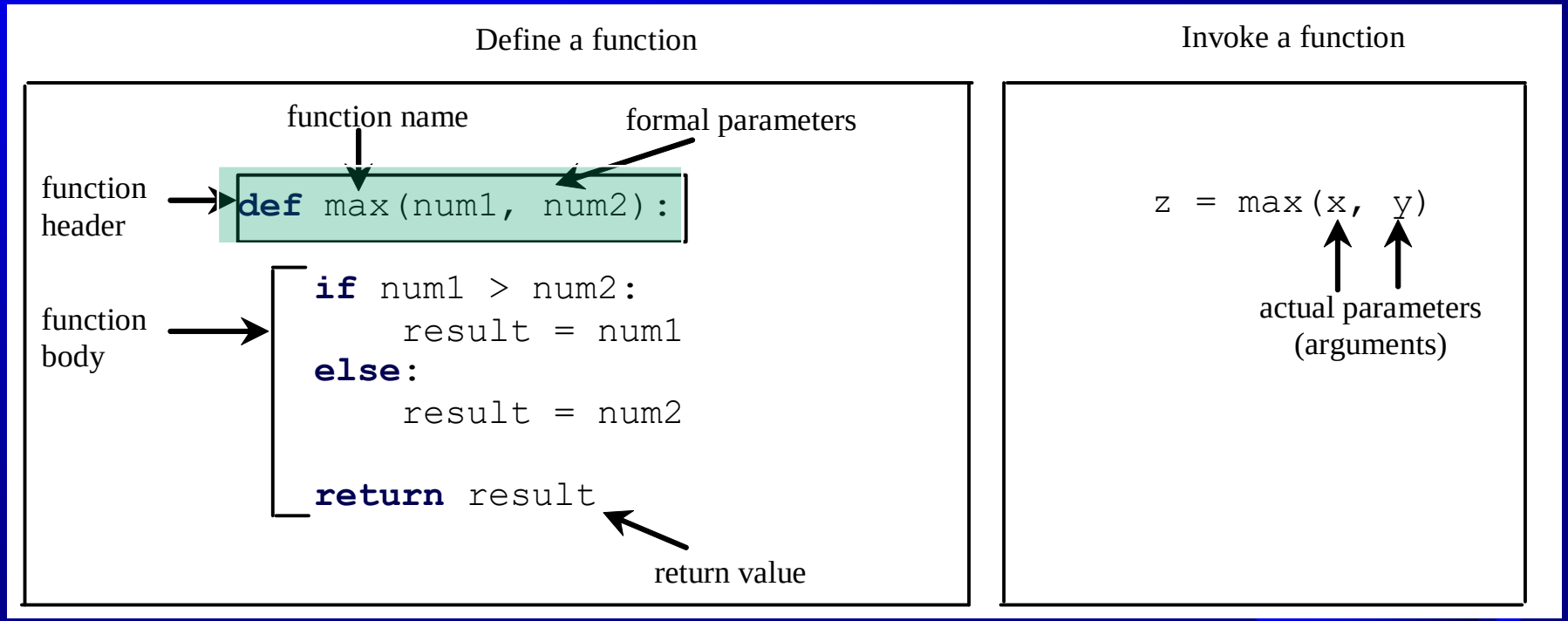
Defining Functions

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



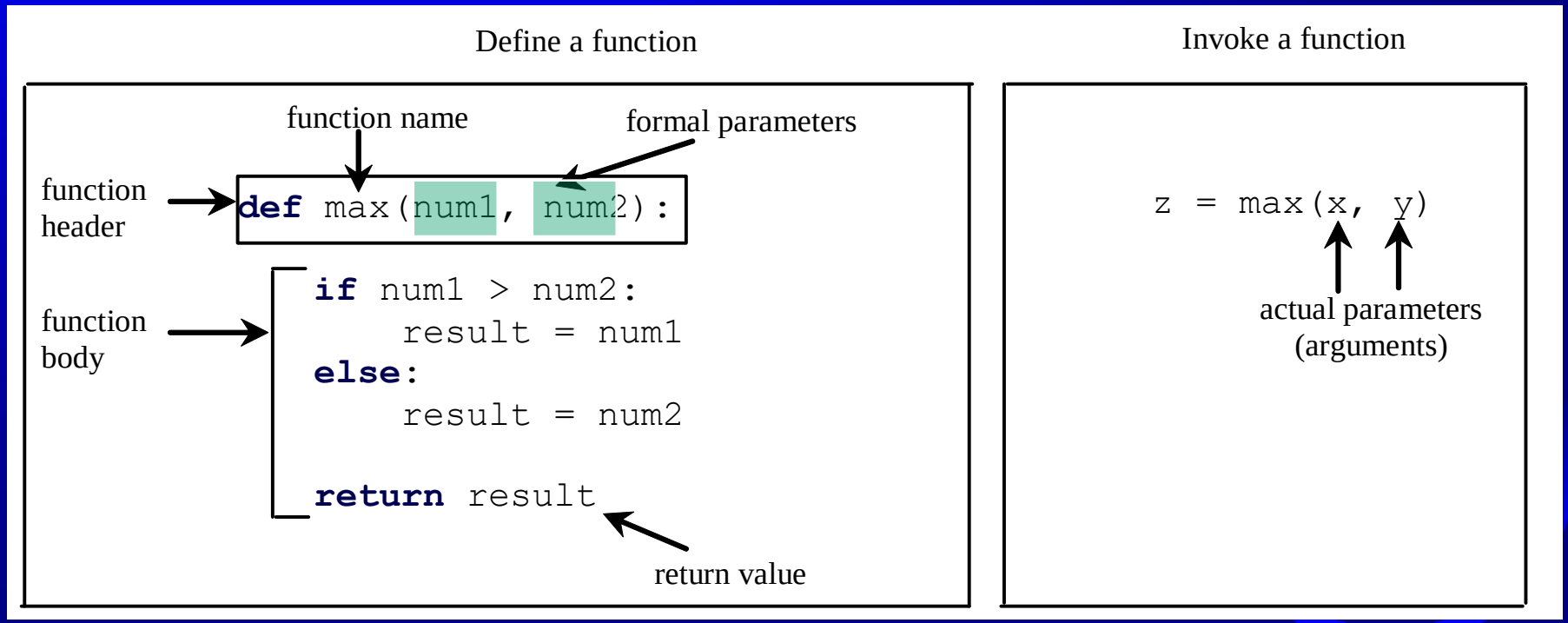
Function Header

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.



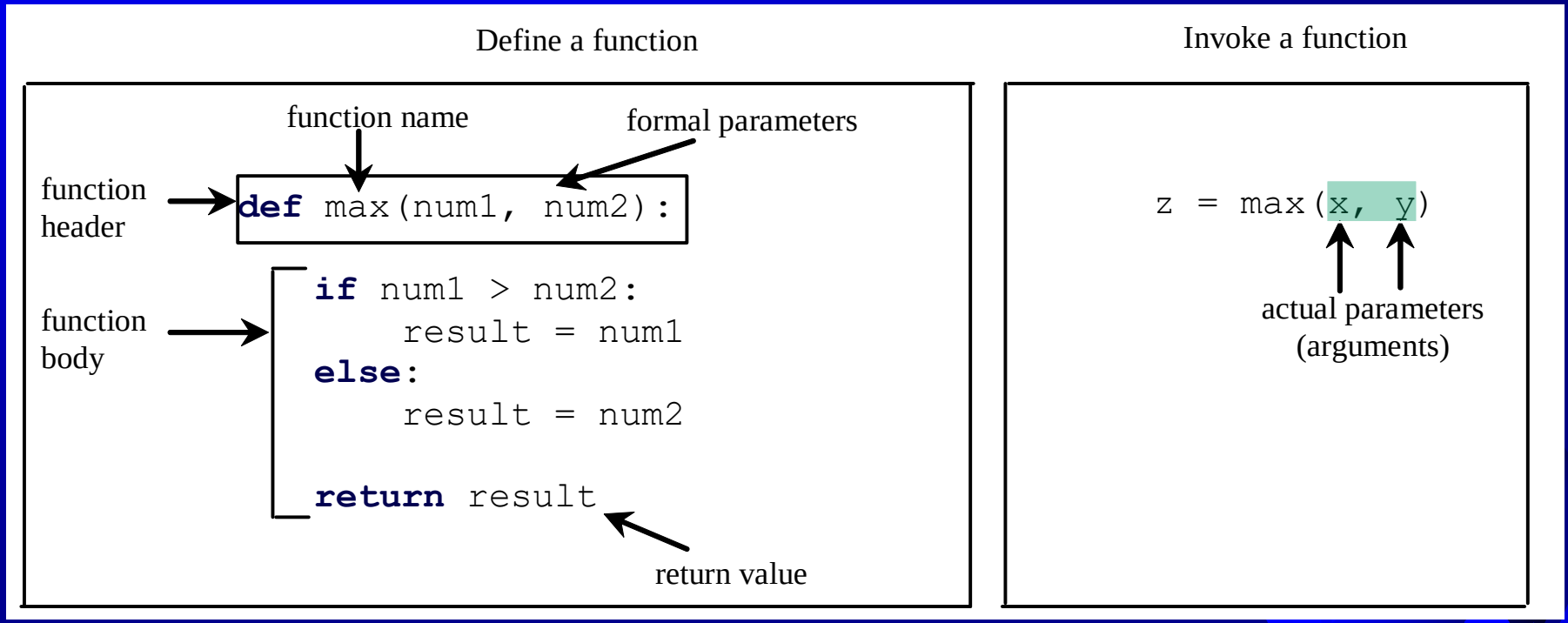
Formal Parameters

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



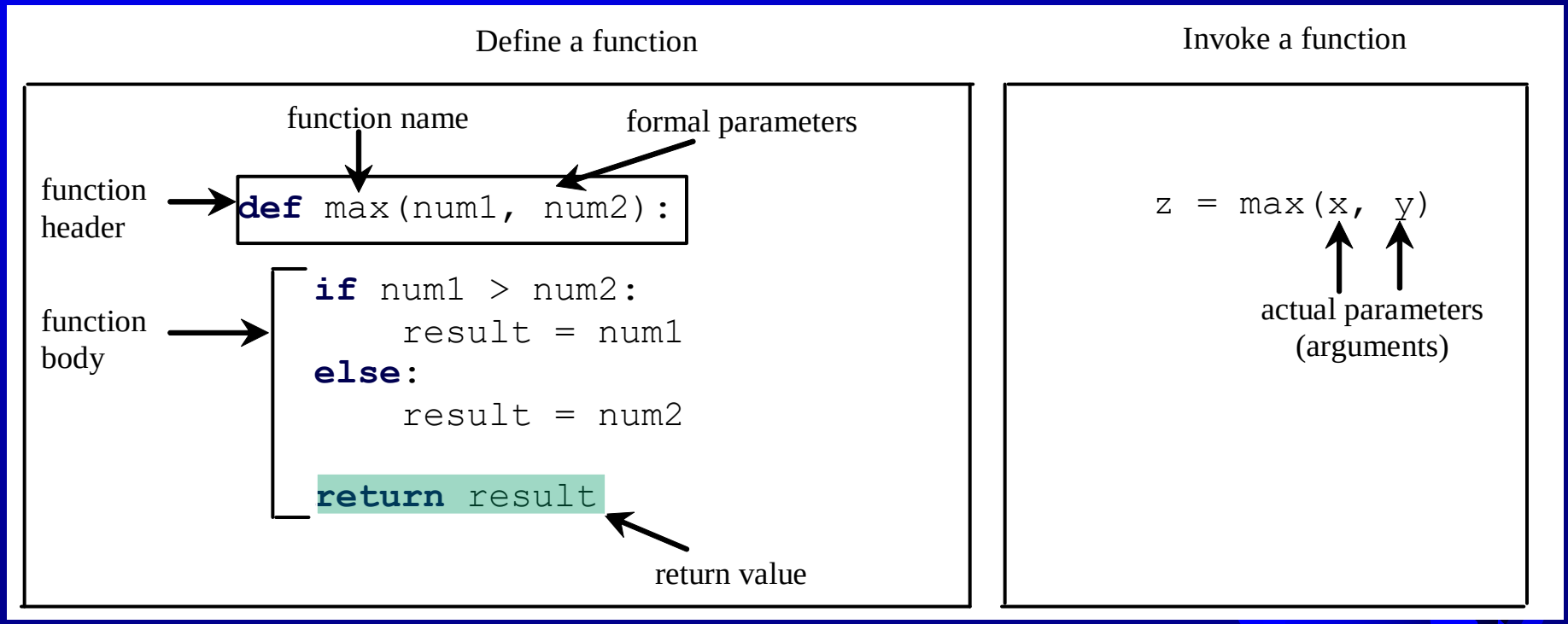
Actual Parameters

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

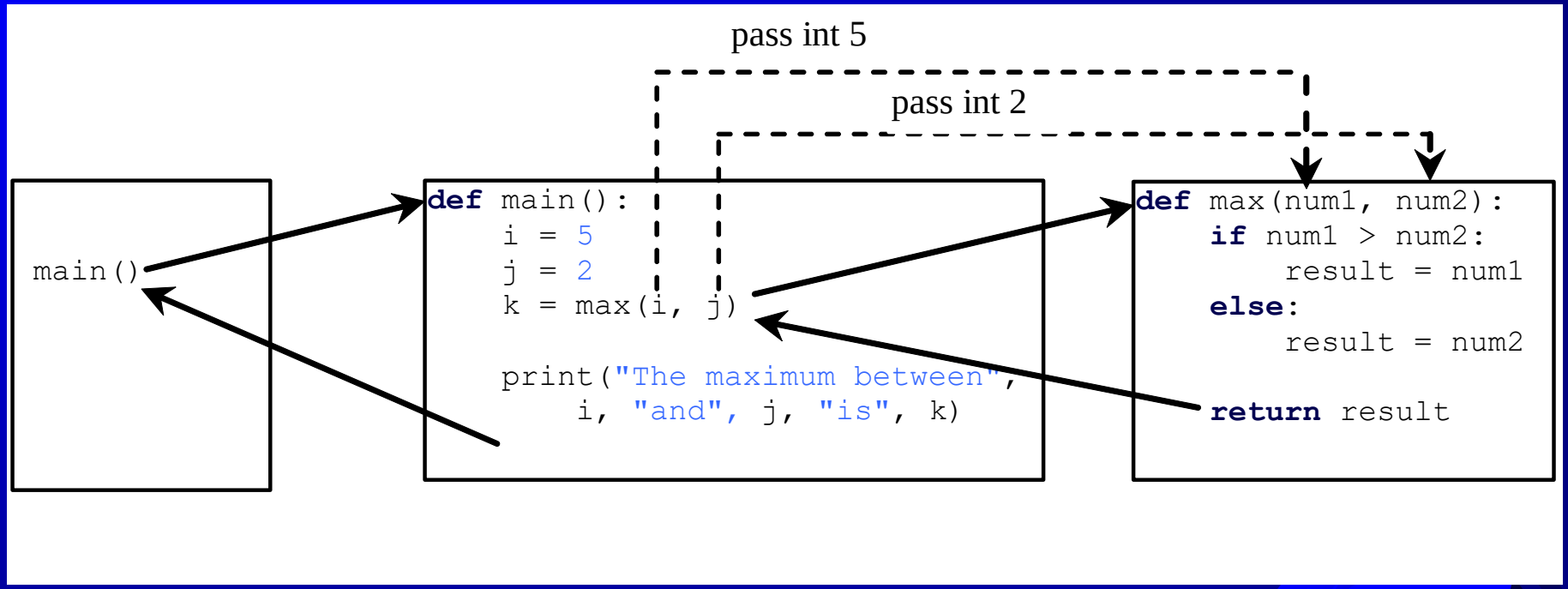
This program demonstrates calling a function max to return the largest of the **`int`** values

TestMax

Run

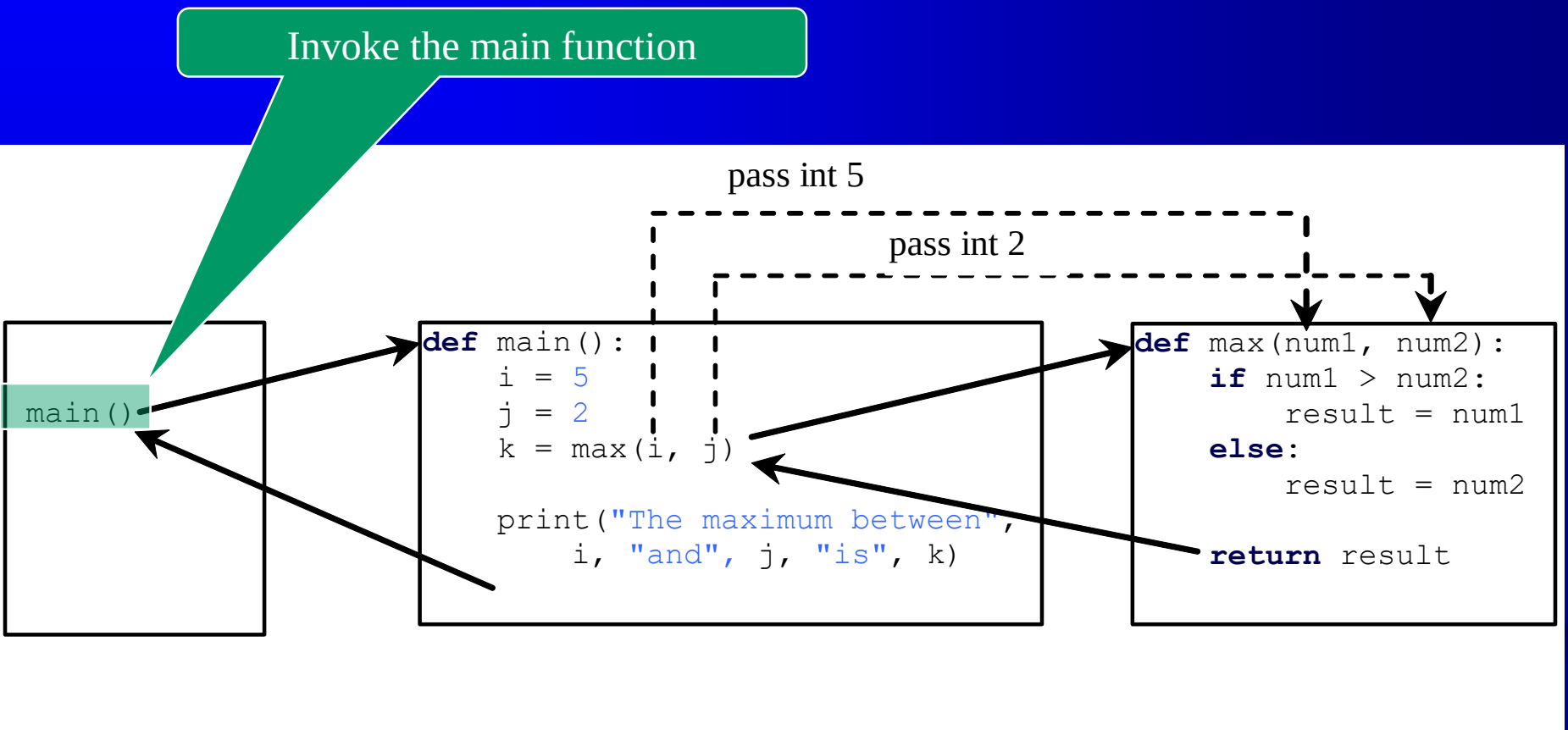
animation

Calling Functions, cont.



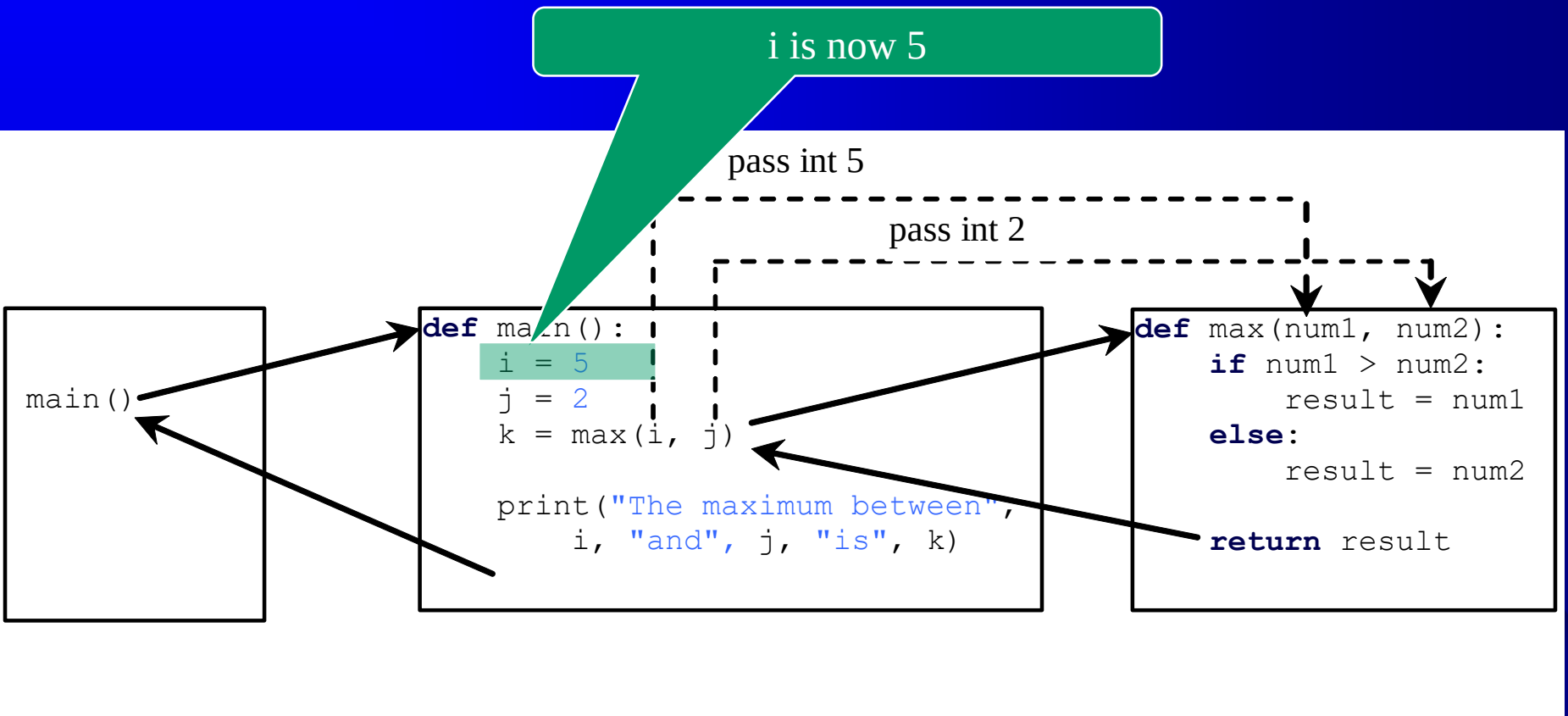
animation

Trace Function Invocation



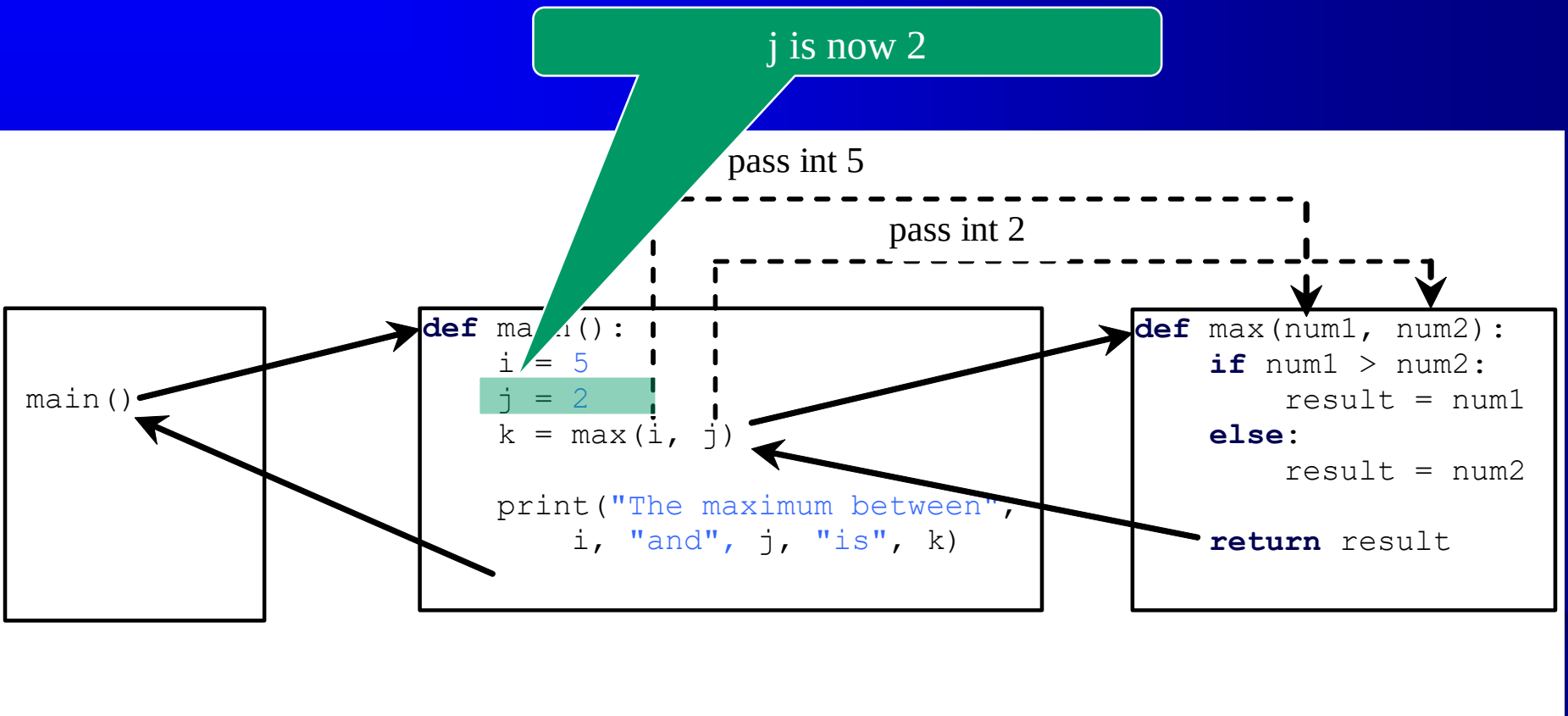
animation

Trace Function Invocation



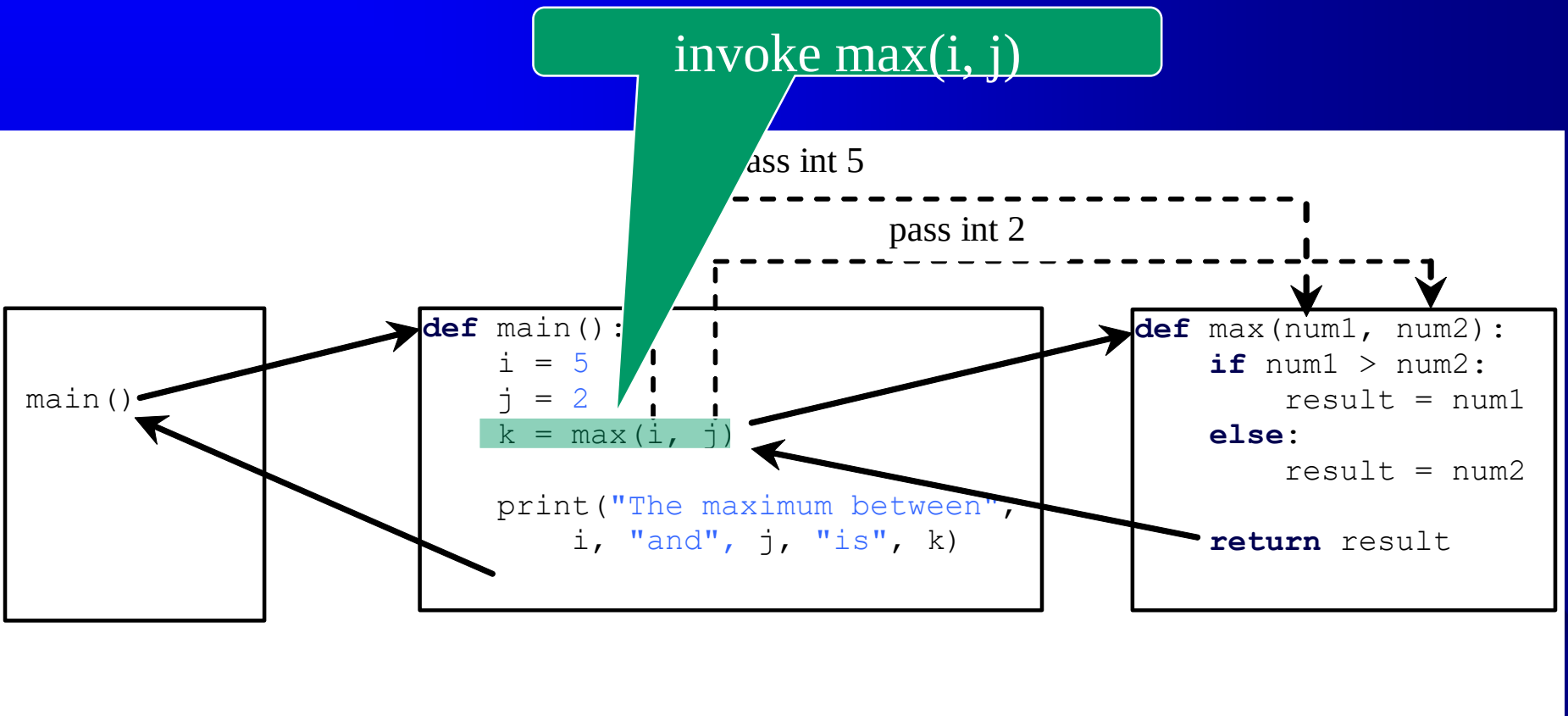
animation

Trace Function Invocation



animation

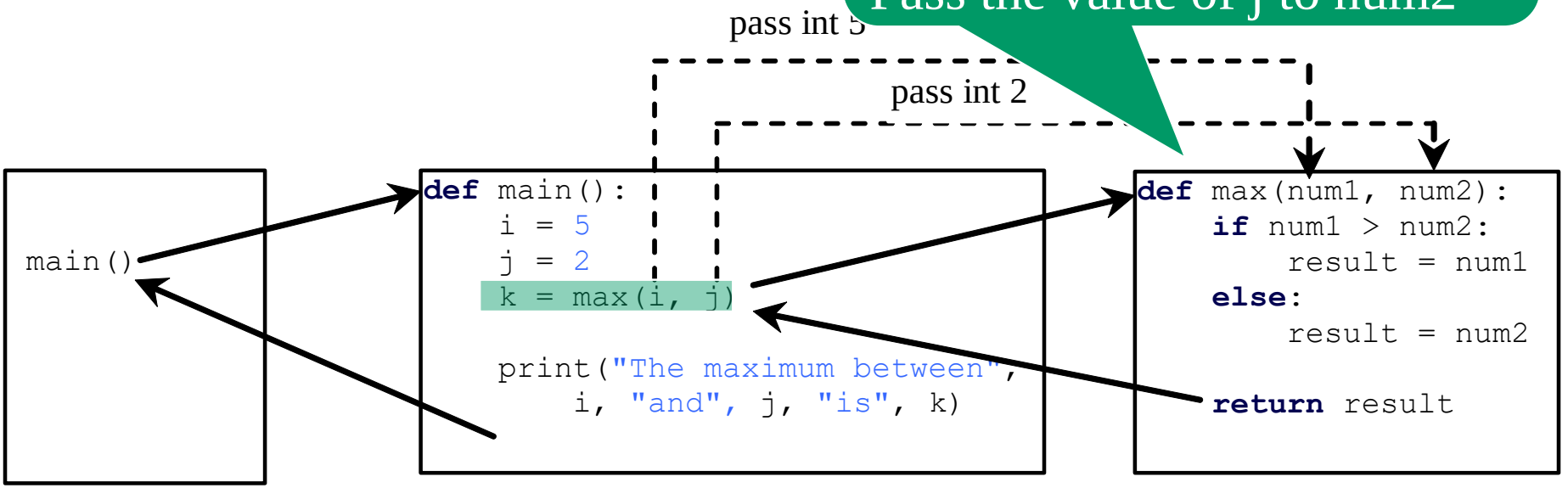
Trace Function Invocation



animation

Trace Function Invocation

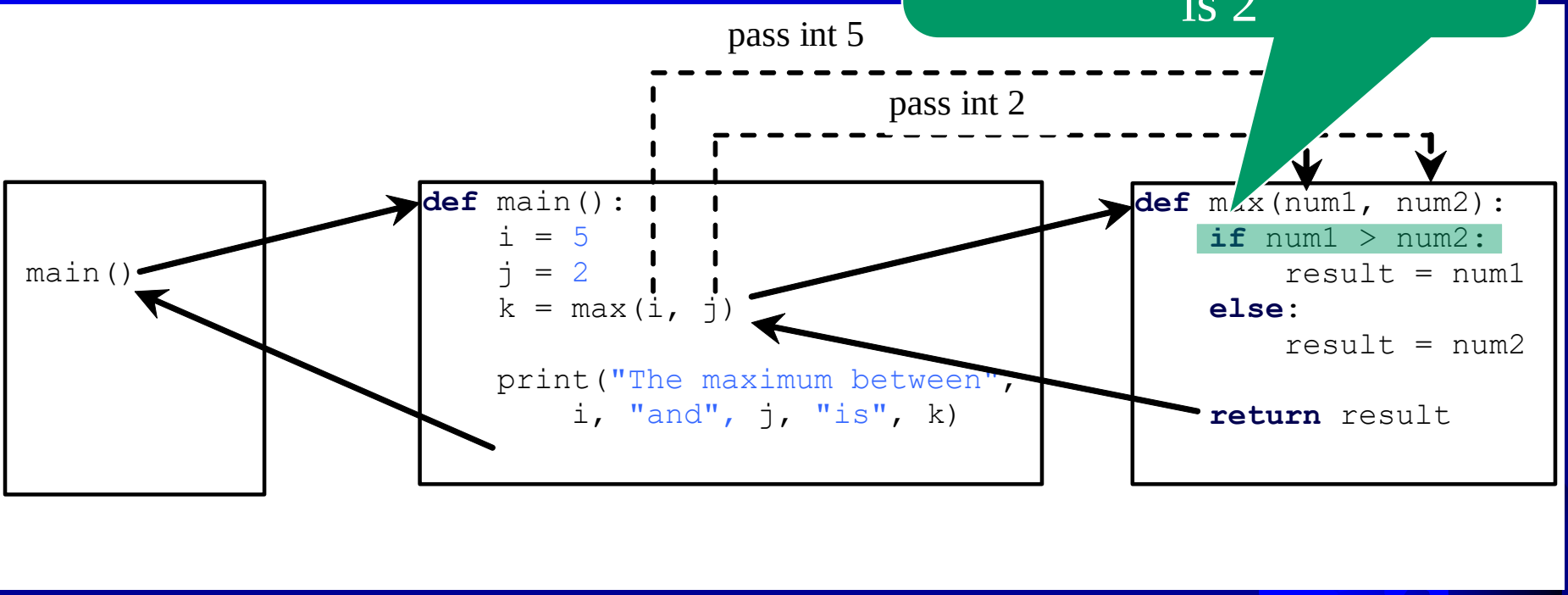
invoke max(i, j)
Pass the value of i to num1
Pass the value of j to num2



animation

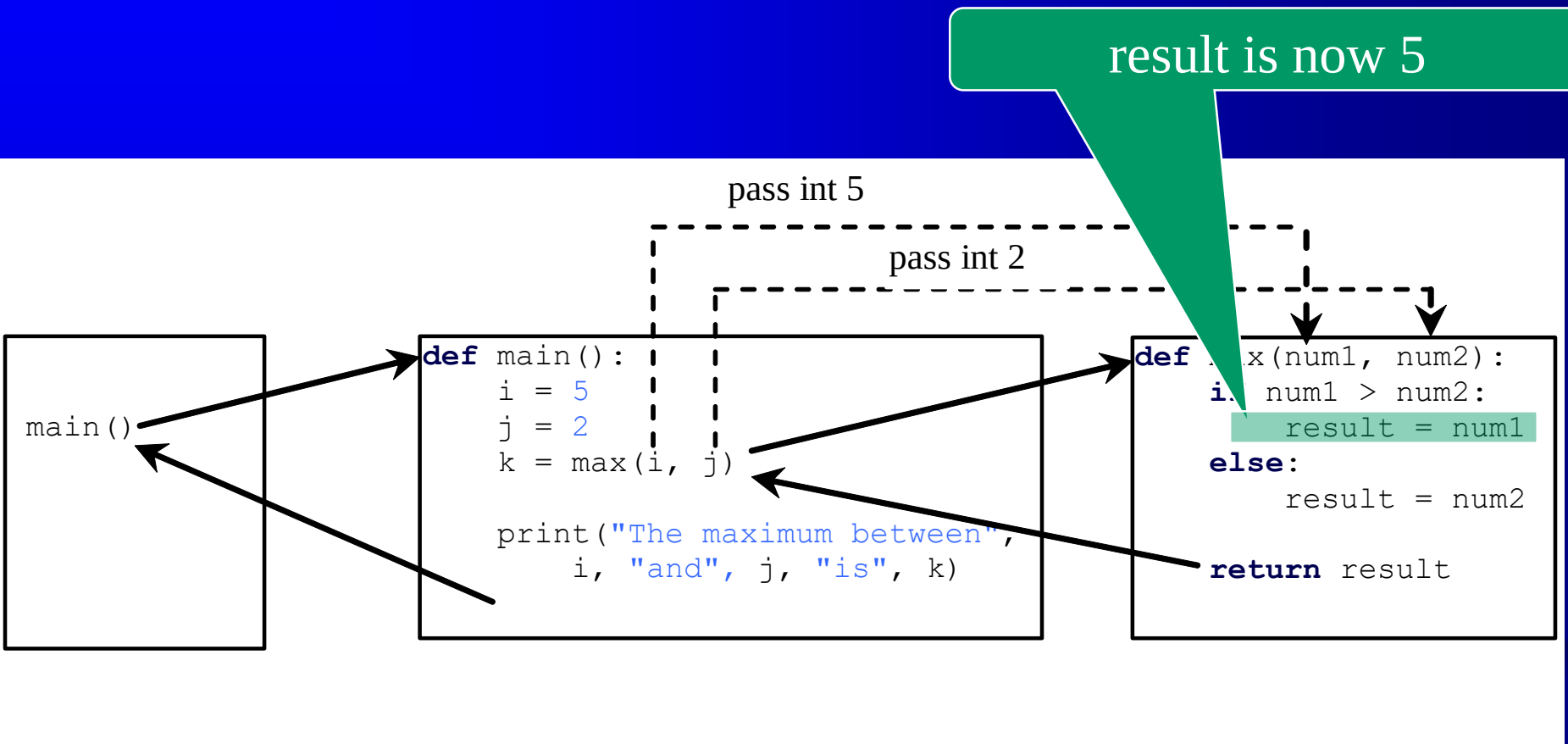
Trace Function Invocation

(num1 > num2) is true since num1 is 5 and num2 is 2



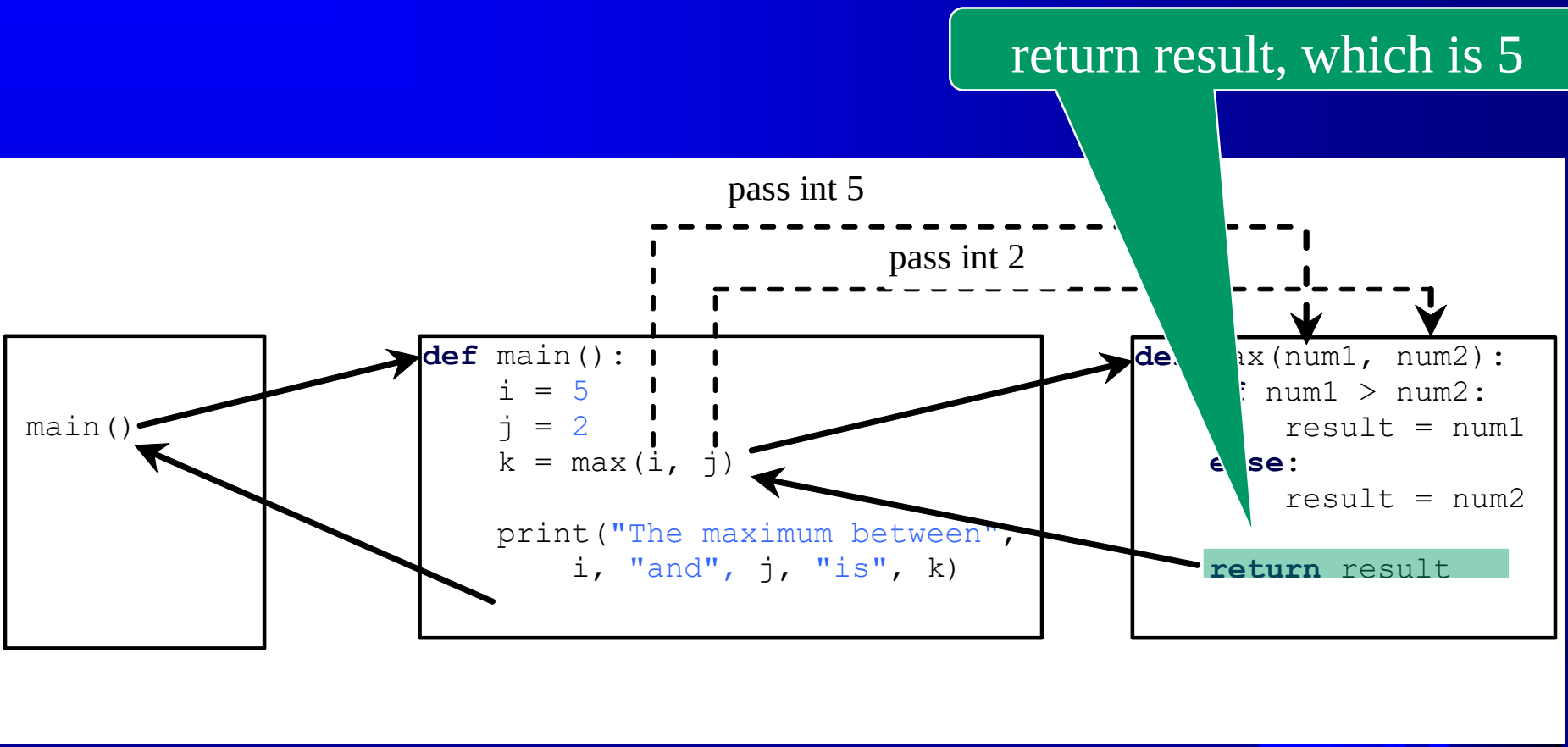
animation

Trace Function Invocation



animation

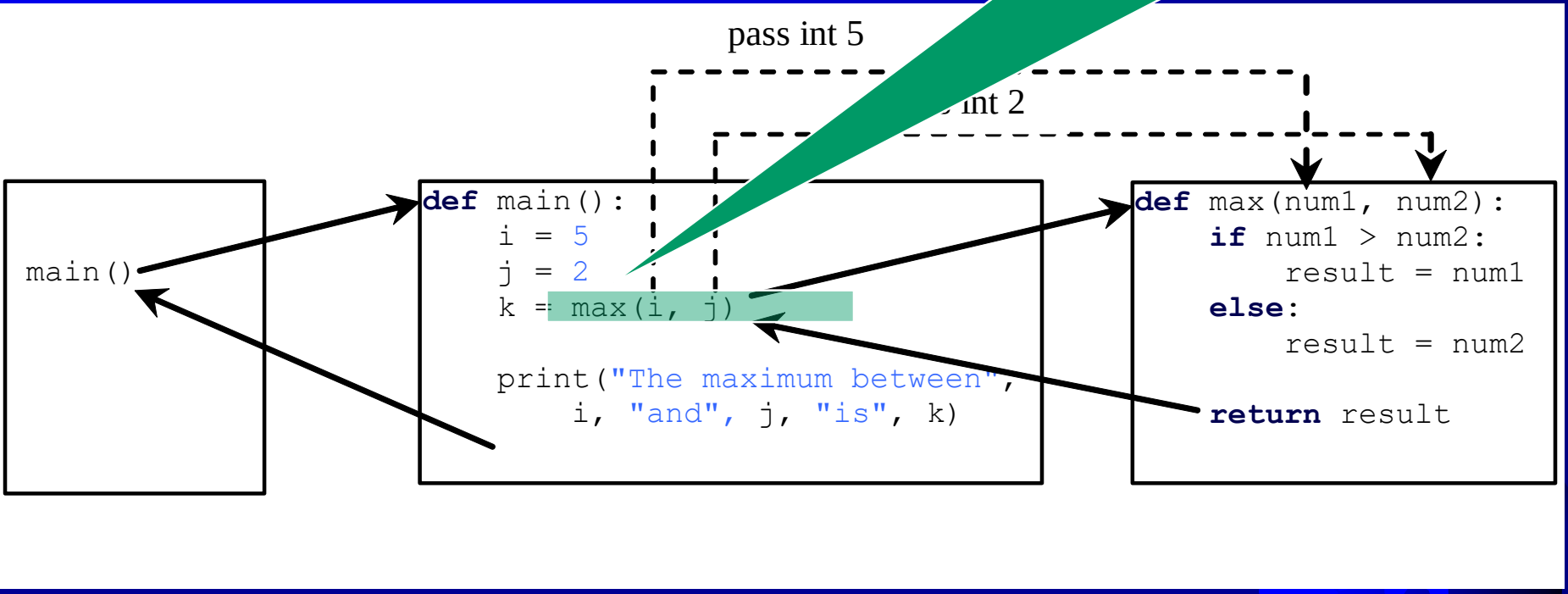
Trace Function Invocation



animation

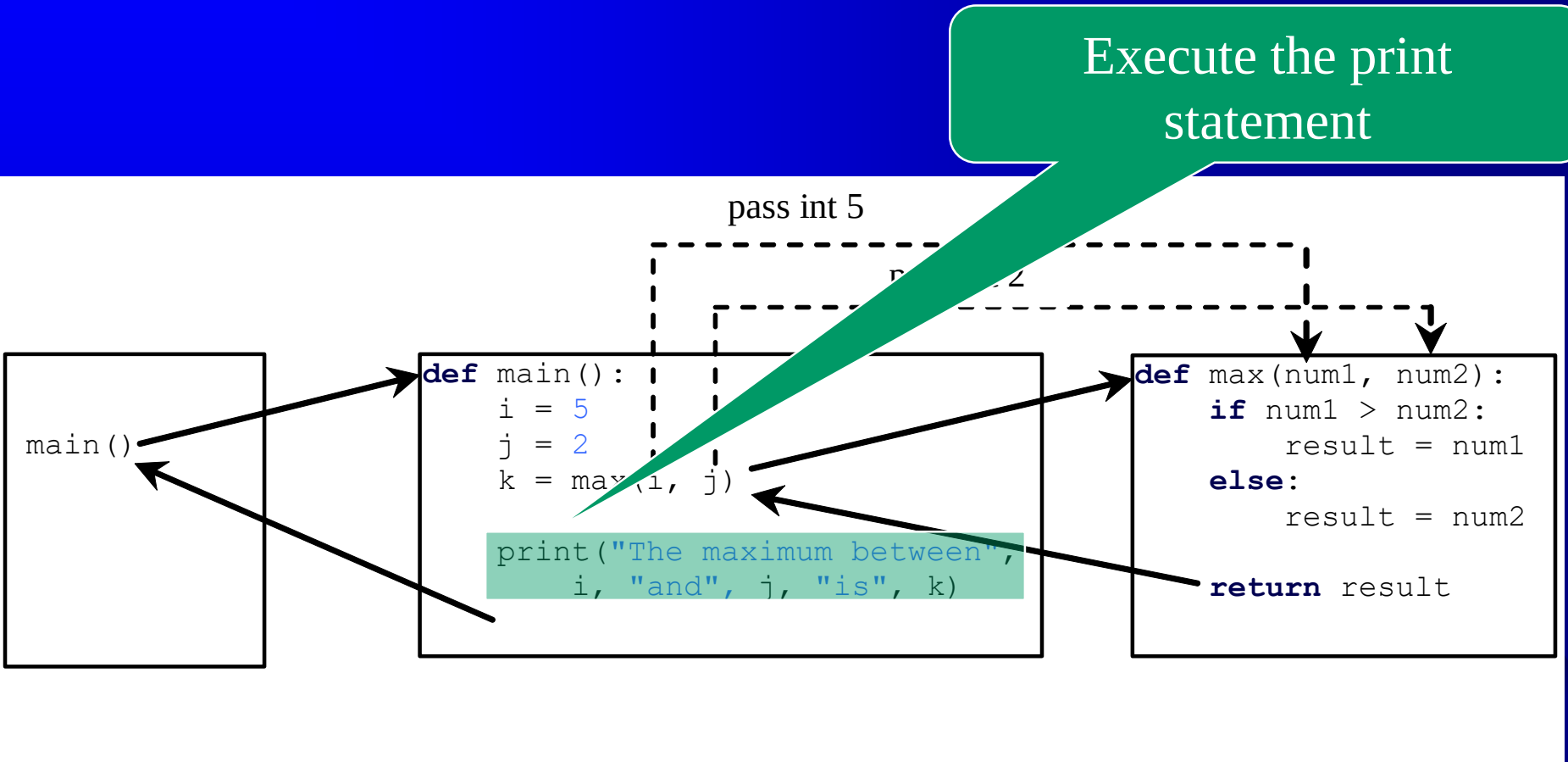
Trace Function Invocation

return max(i, j) and assign the return value to k



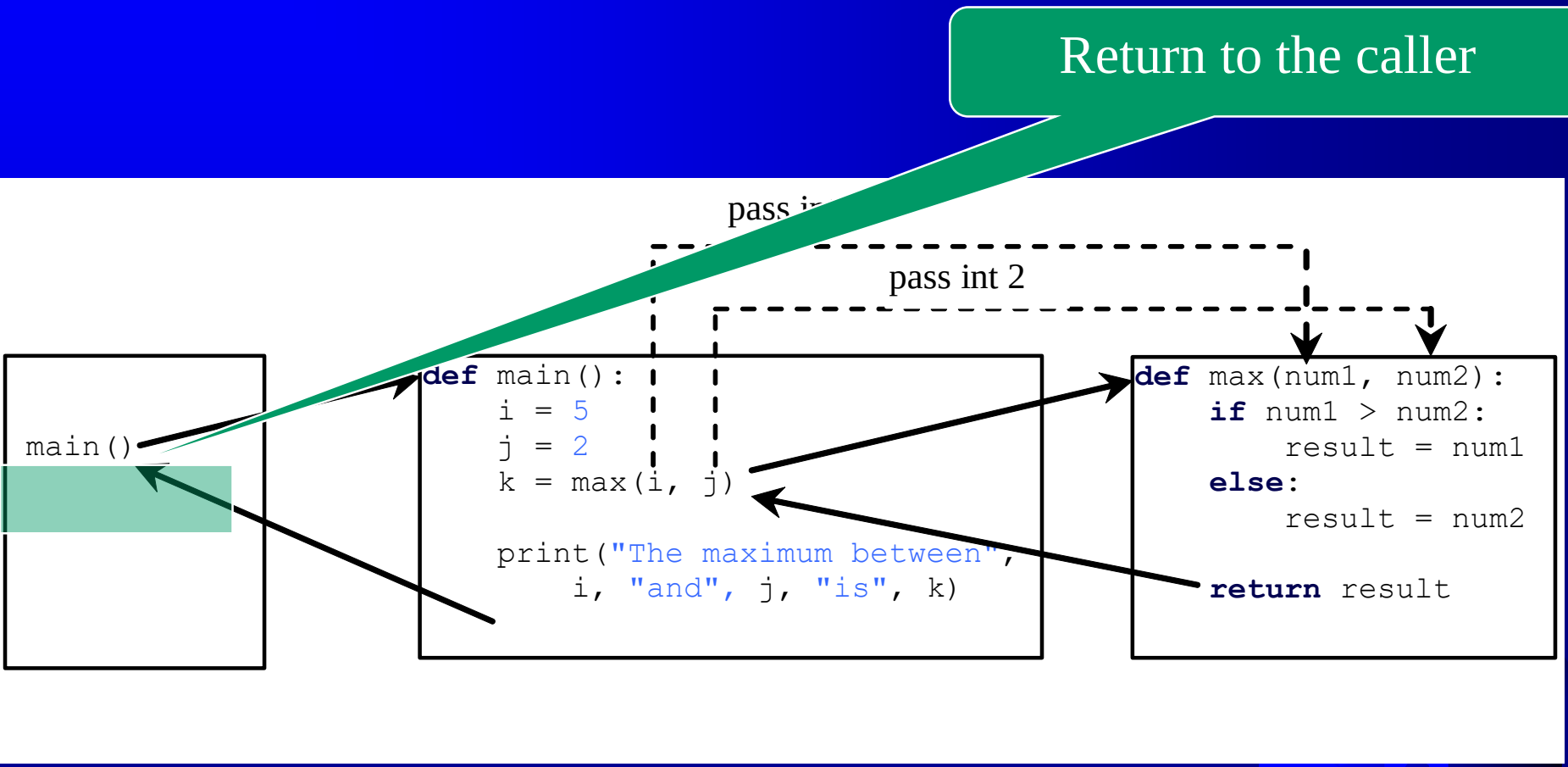
animation

Trace Function Invocation

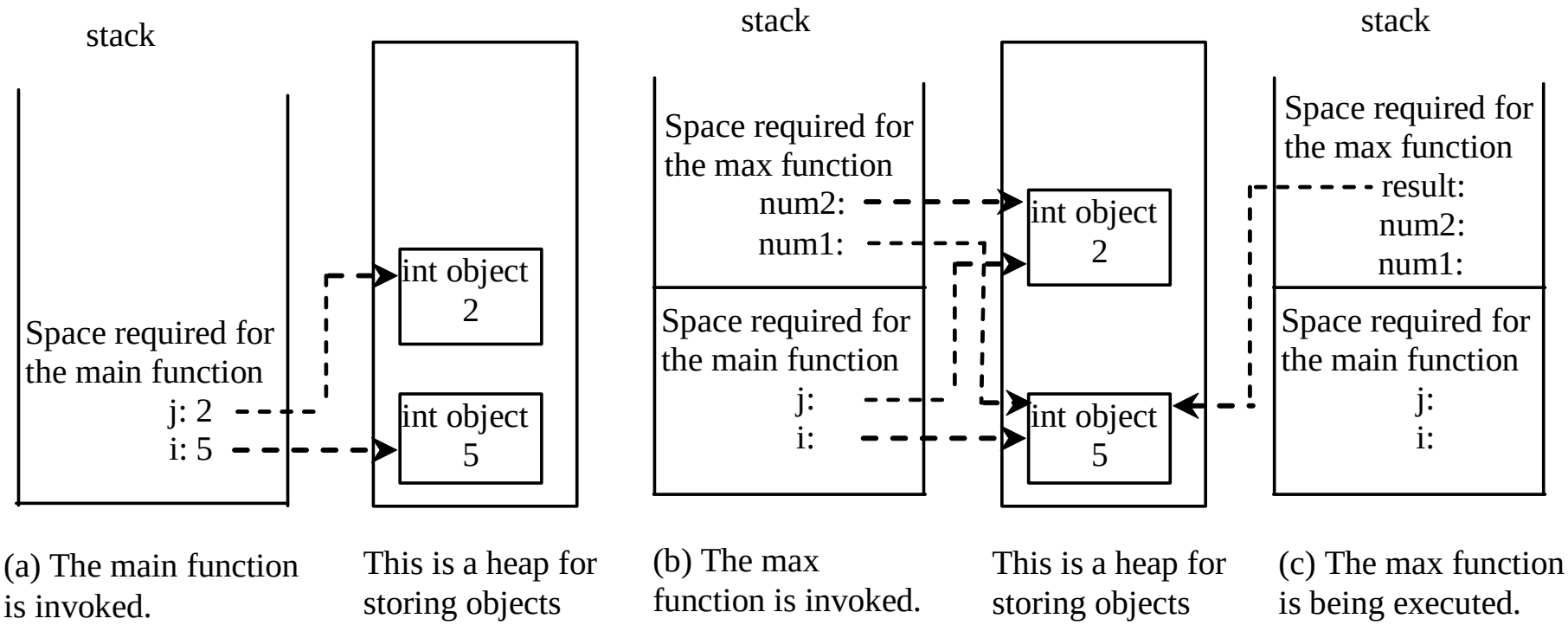


animation

Trace Function Invocation



Call Stacks



(a) The main function is invoked.

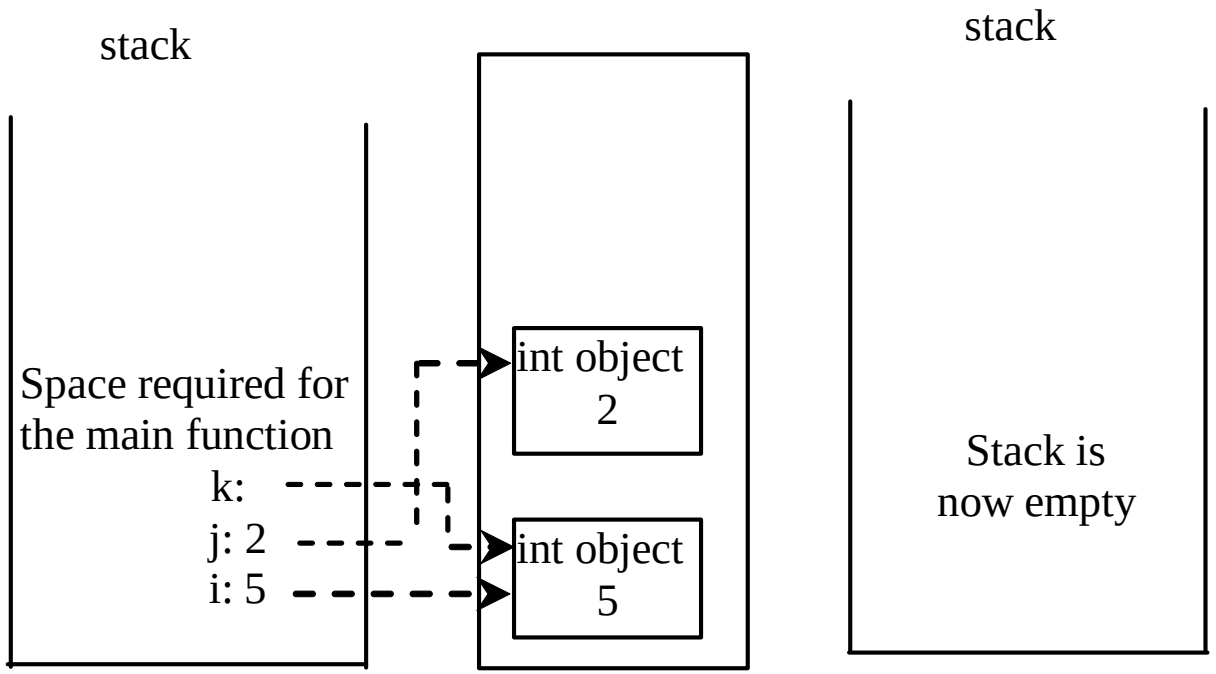
This is a heap for storing objects

(b) The max function is invoked.

This is a heap for storing objects

(c) The max function is being executed.

Call Stacks



(d) The max function is finished and the return value is sent to k.

This is a heap for storing objects

(e) The main function is finished.

Functions With/Without Return Values

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

PrintGradeFunction

Run

ReturnGradeFunction

Run

The None Value

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

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

Passing Arguments by Positions

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

Suppose you invoke the function using

nPrintln("Welcome to Python", 5)

What is the output?

Suppose you invoke the function using

nPrintln("Computer Science", 15)

What is the output?

What is wrong

nPrintln(4, "Computer Science")

Keyword Arguments

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

What is wrong

nPrintln(4, “Computer Science”)

Is this OK?

nPrintln(n = 4, message = “Computer Science”)

Pass by Value

In Python, all data are objects. A variable for an object is actually a reference to the object. When you invoke a function with a parameter, the reference value of the argument is passed to the parameter. This is referred to as *pass-by-value*. For simplicity, we say that the value of an argument is passed to a parameter when invoking a function. Precisely, the value is actually a reference value to the object.

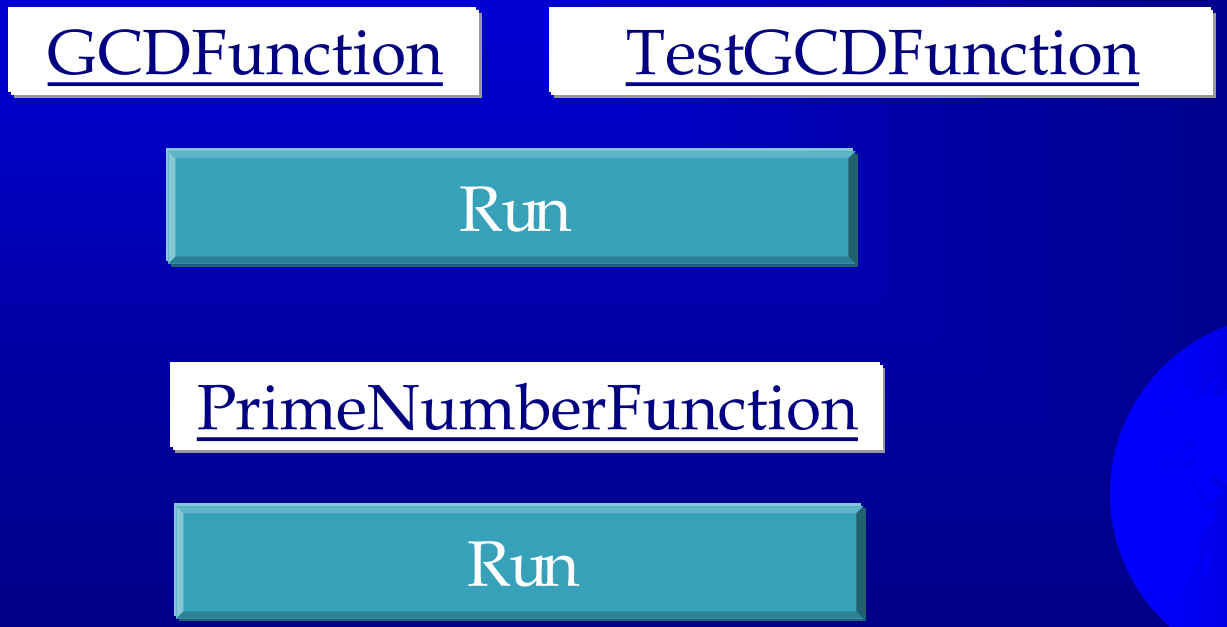
If the argument is a number or a string, the argument is not affected, regardless of the changes made to the parameter inside the function.

Increment

Run

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.



Problem: Converting Decimals to Hexadecimals

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

Decimal2HexConversion

Run

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. This gives an error
```

Example 2

```
x = 1
def f1():
    x = 2
    print(x) # Displays 2
f1()
print(x) # Displays 1
```

Example 3

```
x = eval(input("Enter a number: "))
if (x > 0):
    y = 4
print(y) # This gives an error if y is not created
```

Example 4

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

Example 5

```
x = 1
def increase():
    global x
    x =  x + 1
    print(x) # Displays 2
increase()
print(x) # Displays 2
```

Default Arguments

Python allows you to define functions with default argument values. The default values are passed to the parameters when a function is invoked without the arguments.

DefaultArgumentDemo

Run

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

Run

Generating Random Characters

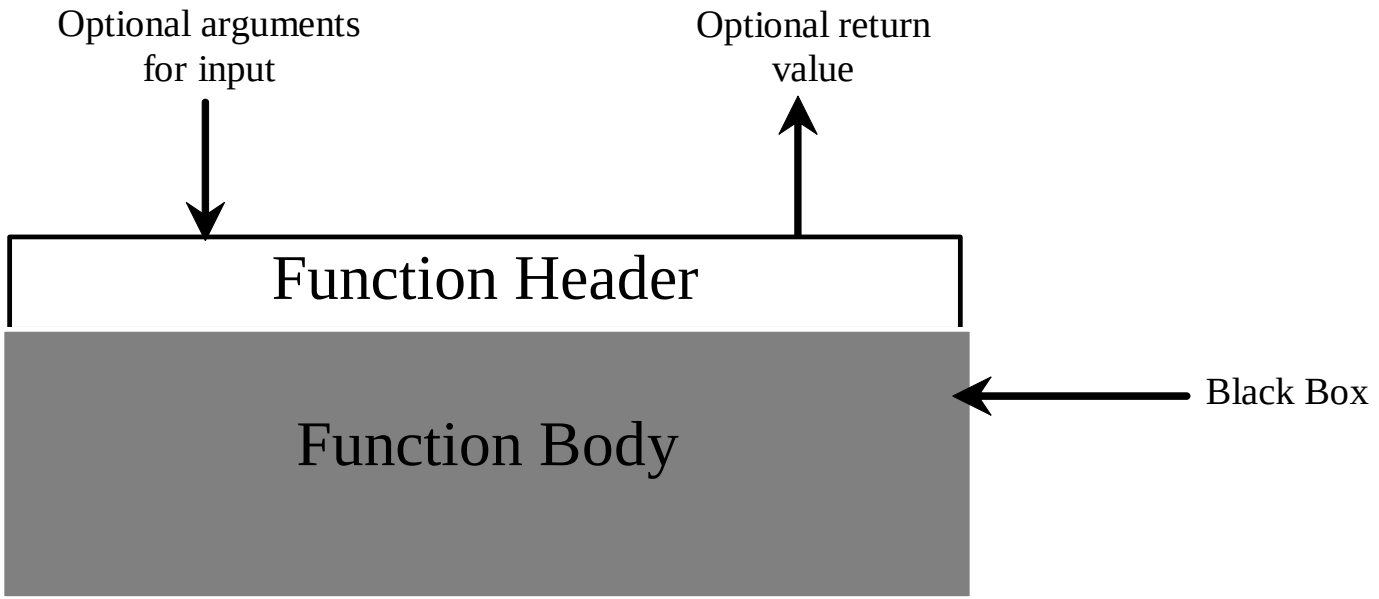
RandomCharacter

TestRandomCharacter

Run

Function Abstraction

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



Benefits of Functions

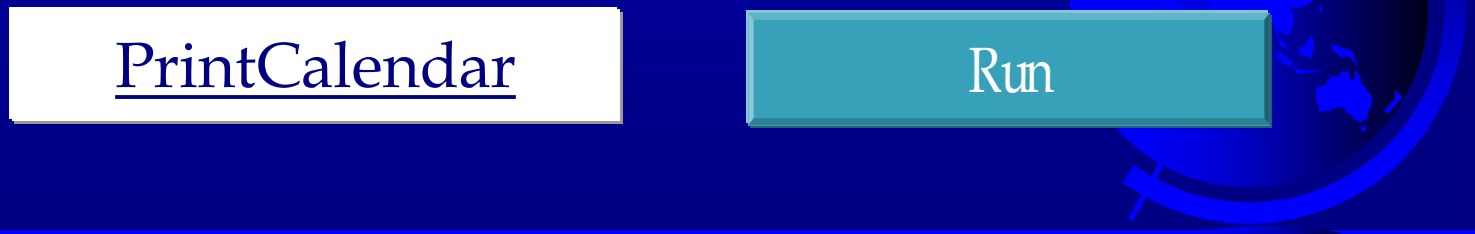
- Write a function once and reuse it anywhere.
- Information hiding. Hide the implementation from the user.
- Reduce complexity.

Stepwise Refinement

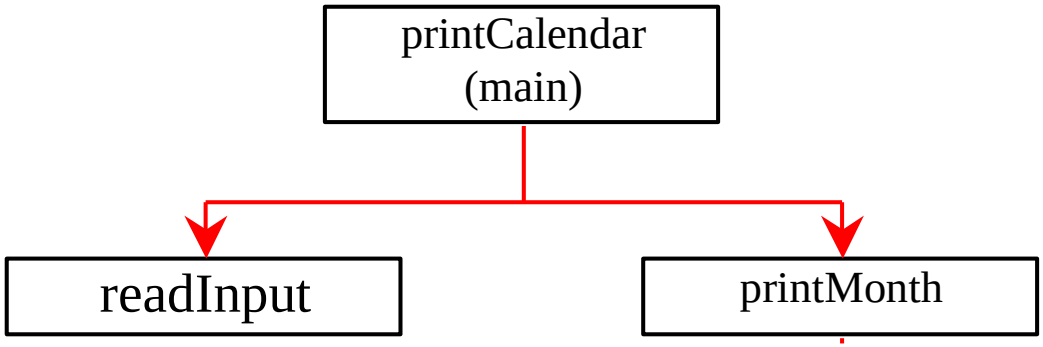
The concept of function abstraction can be applied to the process of developing programs. When writing a large program, you can use the "divide and conquer" strategy, also known as *stepwise refinement*, to decompose it into subproblems. The subproblems can be further decomposed into smaller, more manageable problems.

PrintCalender Case Study

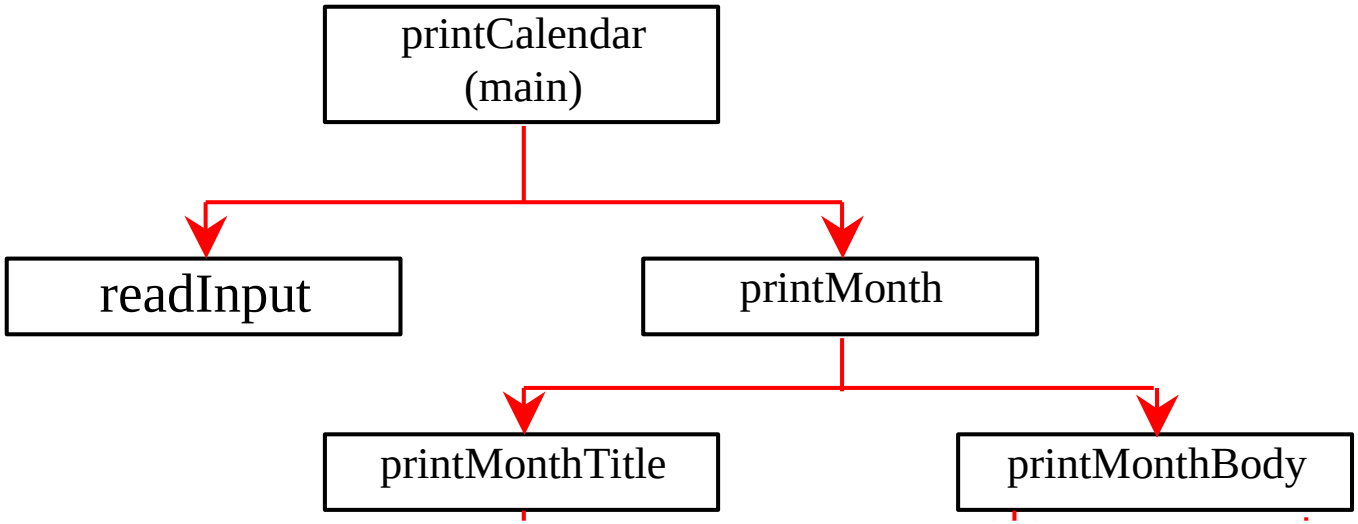
Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.



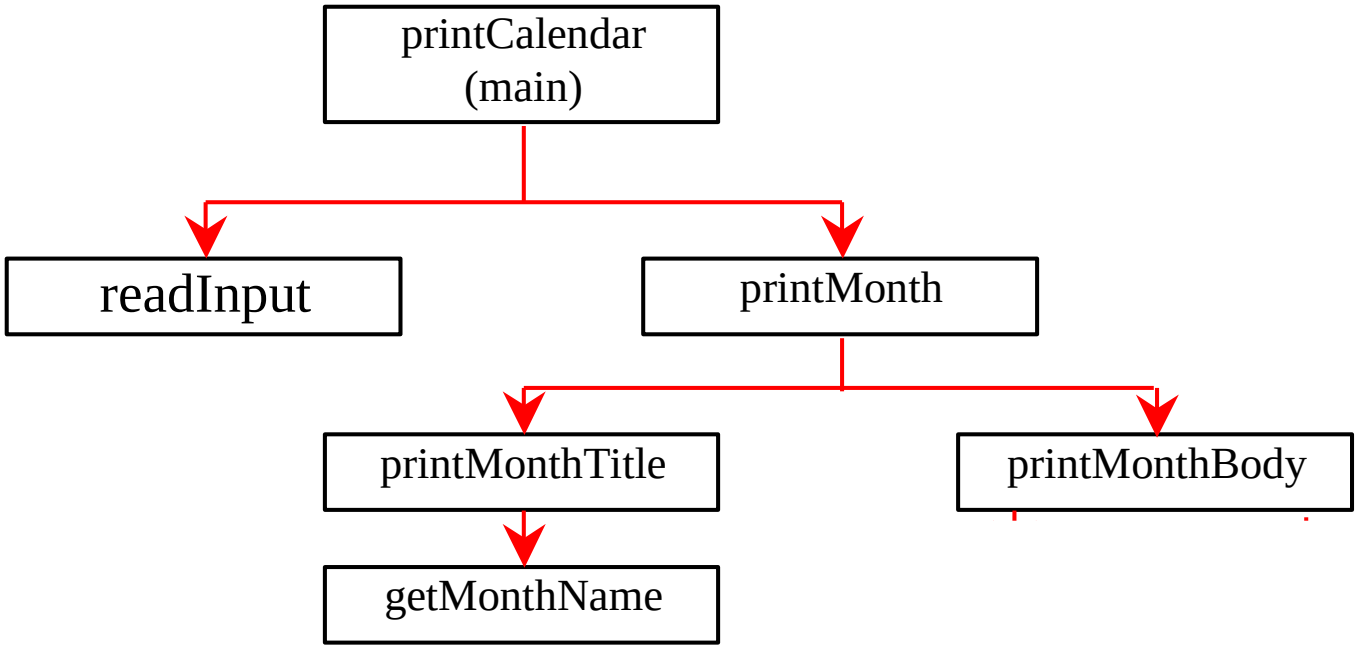
Design Diagram



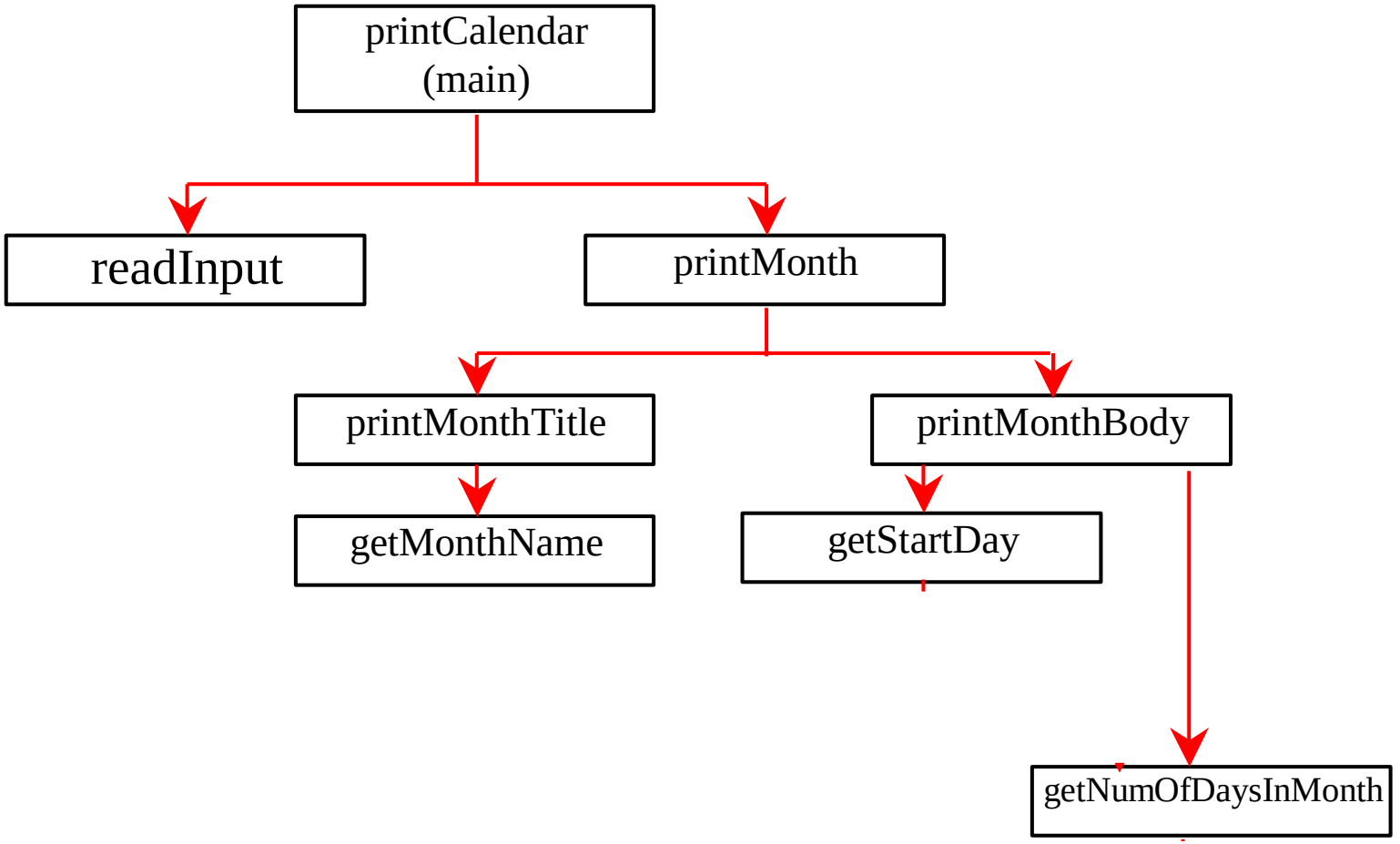
Design Diagram



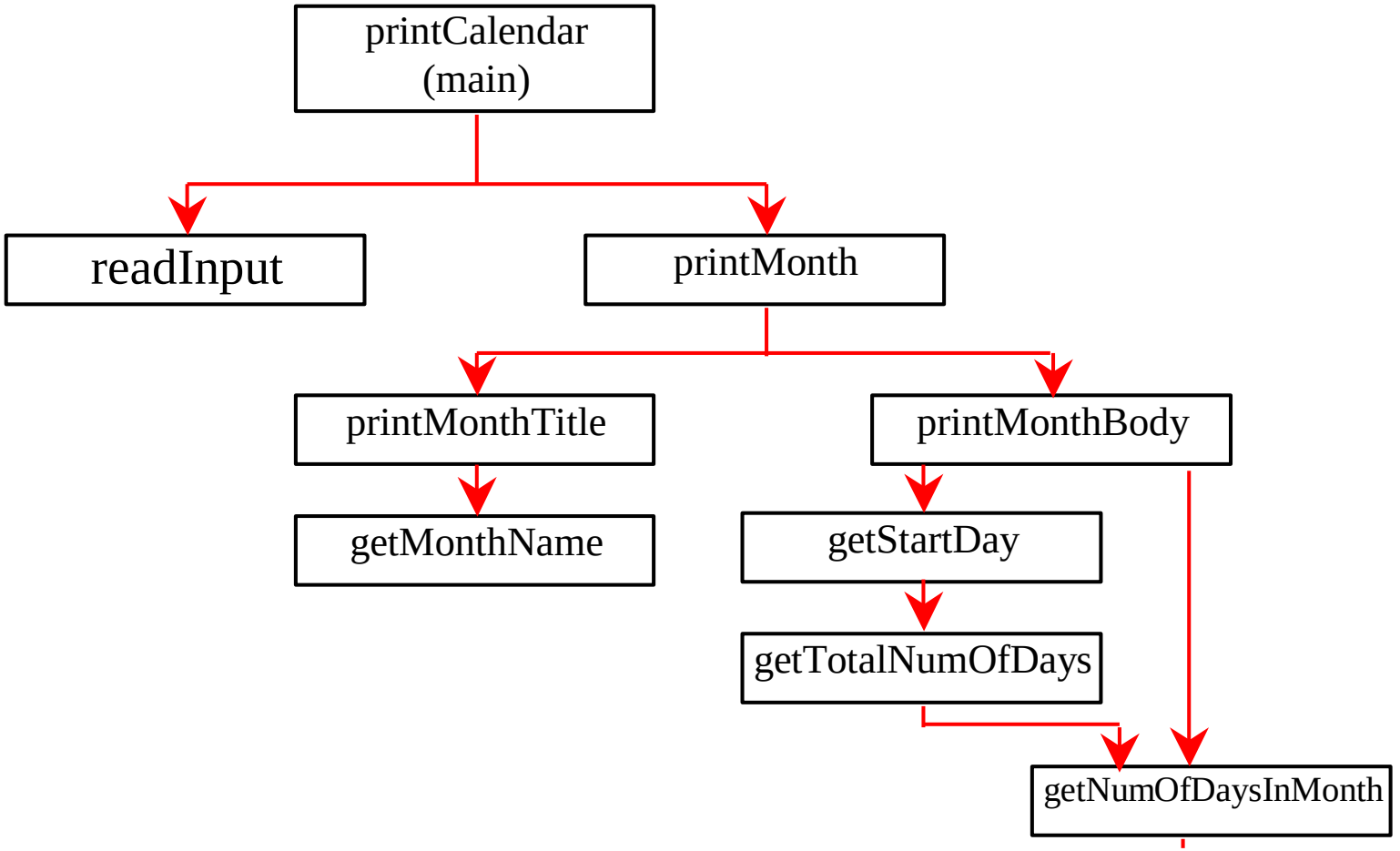
Design Diagram



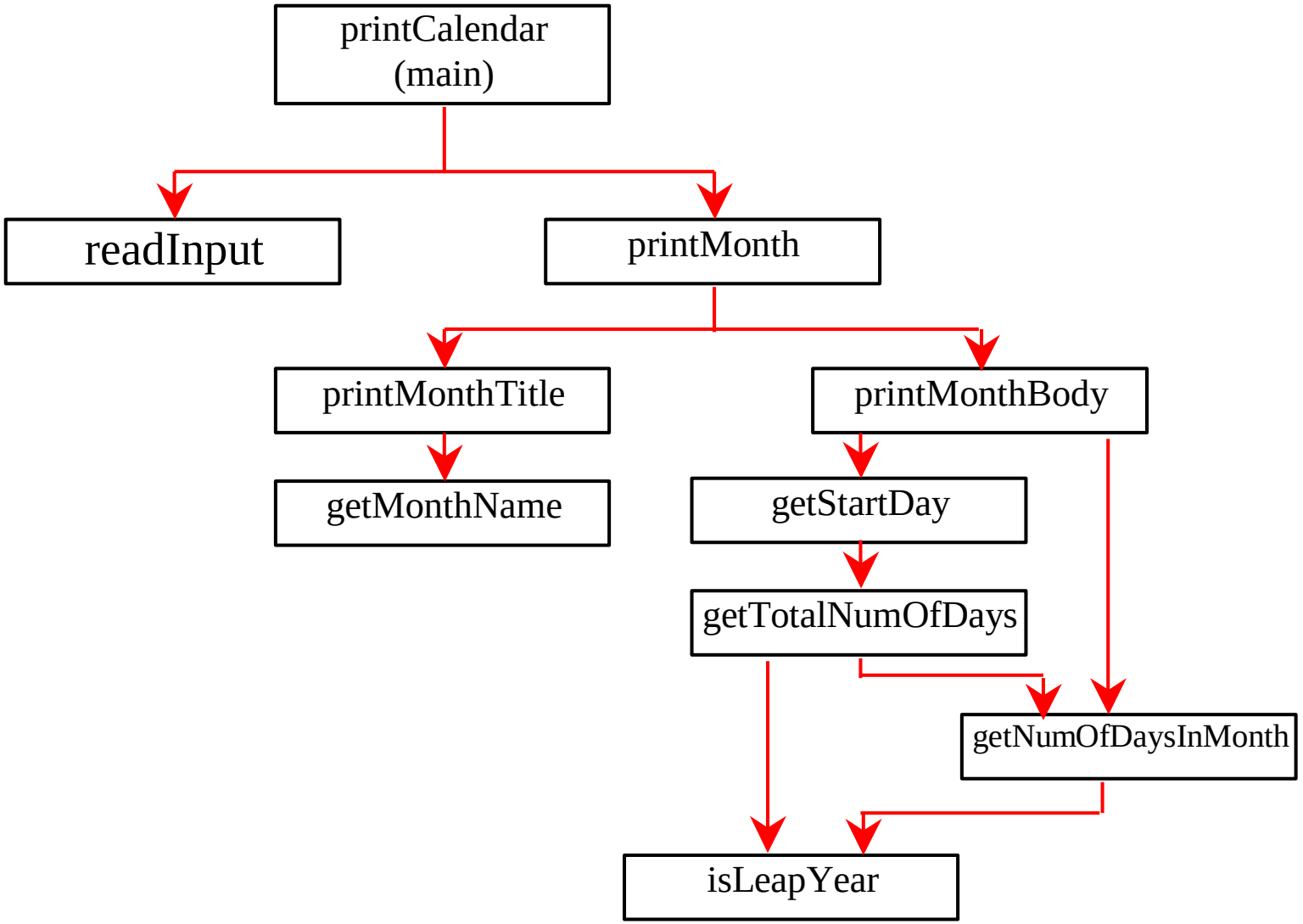
Design Diagram



Design Diagram



Design Diagram



Implementation: Top-Down

Top-down approach is to implement one function in the structure chart at a time from the top to the bottom. Stubs can be used for the functions waiting to be implemented. A stub is a simple but incomplete version of a function. The use of stubs enables you to test invoking the function from a caller. Implement the main function first and then use a stub for the printMonth function. For example, let printMonth display the year and the month in the stub. Thus, your program may begin like this:

A Skeleton for printCalendar

Implementation: Bottom-Up

Bottom-up approach is to implement one function in the structure chart at a time from the bottom to the top. For each function implemented, write a test program to test it. Both top-down and bottom-up functions are fine. Both approaches implement the functions incrementally and help to isolate programming errors and makes debugging easy. Sometimes, they can be used together.

Turtle:
Developing Reusable Graphics Functions

def drawLine(x1, y1, x2, y2):

def writeString(s, x, y):

def drawPoint(x, y):

def drawCircle(x = 0, y = 0, radius = 10):

def drawRectangle(x = 0, y = 0, width = 10, height = 10):

UsefulTurtleFunctions

UseCustomTurtleFunctions

Run