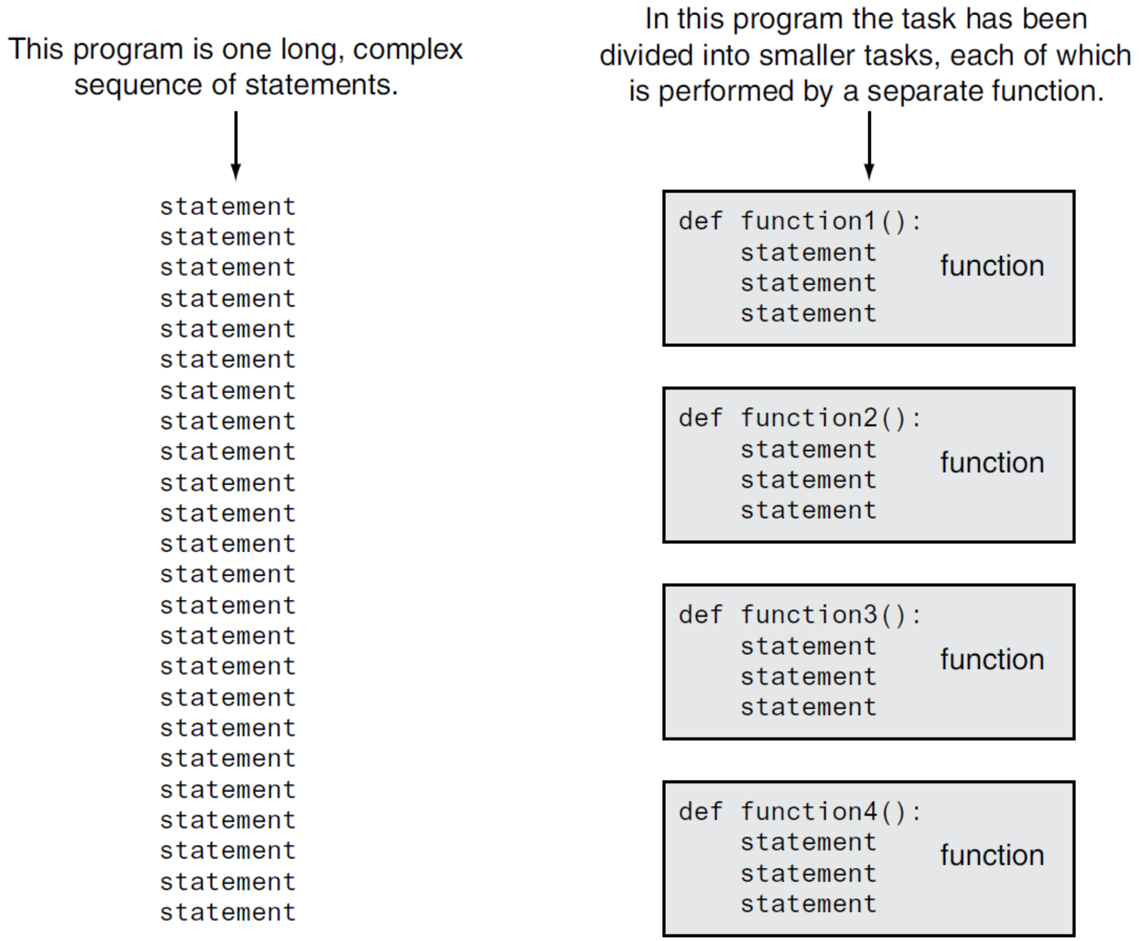
Functions

陳建良

Introduction to Functions

- Most programs perform tasks that are large enough to be broken down into several subtasks.
- For this reason, programmers usually break down their programs into small manageable pieces known as functions.
- A *function* is a group of statements that exist within a program for the purpose of performing a specific task.
- Instead of writing a large program as one long sequence of statements, it can be written as several small functions, each one performing a specific part of the task.
- These small functions can then be executed in the desired order to perform the overall task.

- This approach is sometimes called ***divide and conquer*** because a large task is divided into several smaller tasks that are easily performed.


This program is one long, complex sequence of statements.
In this program the task has been divided into smaller tasks, each of which is performed by a separate function.
function
function
function
function

```
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
```

```
def function1():
    statement
    statement
    statement
```

```
def function2():
    statement
    statement
    statement
```

```
def function3():
    statement
    statement
    statement
```

```
def function4():
    statement
    statement
    statement
```

Benefits of Modularizing a Program with Functions

- Simpler Code
- Code Reuse
- Better Testing
- Faster Development
- Easier Facilitation of Teamwork

Void Functions and Value-returning Functions

- There are two types of functions: void functions and value returning functions.
- When you call a ***void function***, it simply executes the statements it contains and then terminates.
- When you call a ***value-returning function***, it executes the statements that it contains, then returns a value back to the statement that called it.
- The input function is an example of a value-returning function.

Function Names

- Just as you name the variables that you use in a program, you also name the functions.
- Python requires that you follow the same rules that you follow when naming variables, which we recap here:
  - You cannot use one of Python's key words as a function name.
  - A function name cannot contain spaces.
  - The first character must be one of the letters a through z, A through Z, or an underscore character (_).
  - After the first character you may use the letters a through z or A through Z, the digits 0 through 9, or underscores.
  - Uppercase and lowercase characters are distinct.

- Here is the general format of a function definition in Python:

```
def function_name():
    statement
    statement
    etc.
```

- An example of a function.

```
def message():
    print('I am Arthur,')
    print('King of the Britons.')
```

- To execute a function, you must *call* it.

```
message()
```

```python
# This program demonstrates a function.
# First, we define a function named message.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the message function.
message()
```

Program Output

```
I am Arthur,
King of the Britons.
```

These statements cause the message function to be created.

```
# This program demonstrates a function.
# First, we define a function named message.
def message():
    print('I an Arthur,')
    print('King of the Britons.')

# Call the message function.
message()
```

This statement calls the message function, causing it to execute.

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
     print (x)

print ("Goodbye!")
```

Output

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
     print (x)

print ("Goodbye!")
```

Output

```
Welcome!
```

Aletheia University 資訊工程學系

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
     print (x)

print ("Goodbye!")
```

Output

```
Welcome!
```

Aletheia University 真理大學 資訊工程學系

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
      print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
```

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
     print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
```

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
      print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
1
```

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
      print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
1
```

Aletheia University
真理大學
資訊工程學系

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
      print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
1
2
```

Flow of Execution

Code

```
print ("Welcome!")

for x in range(3):
     print (x)

print ("Goodbye!")
```

Output

```
Welcome!
0
1
2
Goodbye!
```

Flow of Functions with Functions

Code

```
def hello():
     print ("Hi there!")
     print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

Flow of Functions with Functions

Code

Output

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Flow of Functions with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
```

Flow of Functions with Functions

Code

```
def hello():
     print ("Hi there!")
     print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Functions with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Functions with Functions

Code

```
def hello():
      print ("Hi there!")
      print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
```

Flow of Functions with Functions

Code

```
def hello():
    print ("Hi there!")
    print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
```

Flow of Functions with Functions

Code

```
def hello():
     print ("Hi there!")
     print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

真理大學 Aletheia University 資訊工程學系

Flow of Functions with Functions

Code

```
def hello():
     print ("Hi there!")
     print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
```

Aletheia University 資訊工程學系

Flow of Functions with Functions

Code

```
def hello():
     print ("Hi there!")
     print ("I'm a function!")

print ("Good morning")
print ("Welcome to class")

hello()

print ("And now we're done.")
```

Output

```
Good morning
Welcome to class
Hi there!
I'm a function!
And now we're done.
```

Multiple Functions

```
def hello():
    print (“Hello there!”)
def goodbye():
    print (“See ya!”)
hello()
goodbye()
```

Multiple Functions

- In fact, it is common for a program to have a main function that is called when the program starts.
- The main function then calls other functions in the program as they are needed.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

Program Output

```
I have a message for you.
I am Arthur,
King of the Britons.
Goodbye!
```

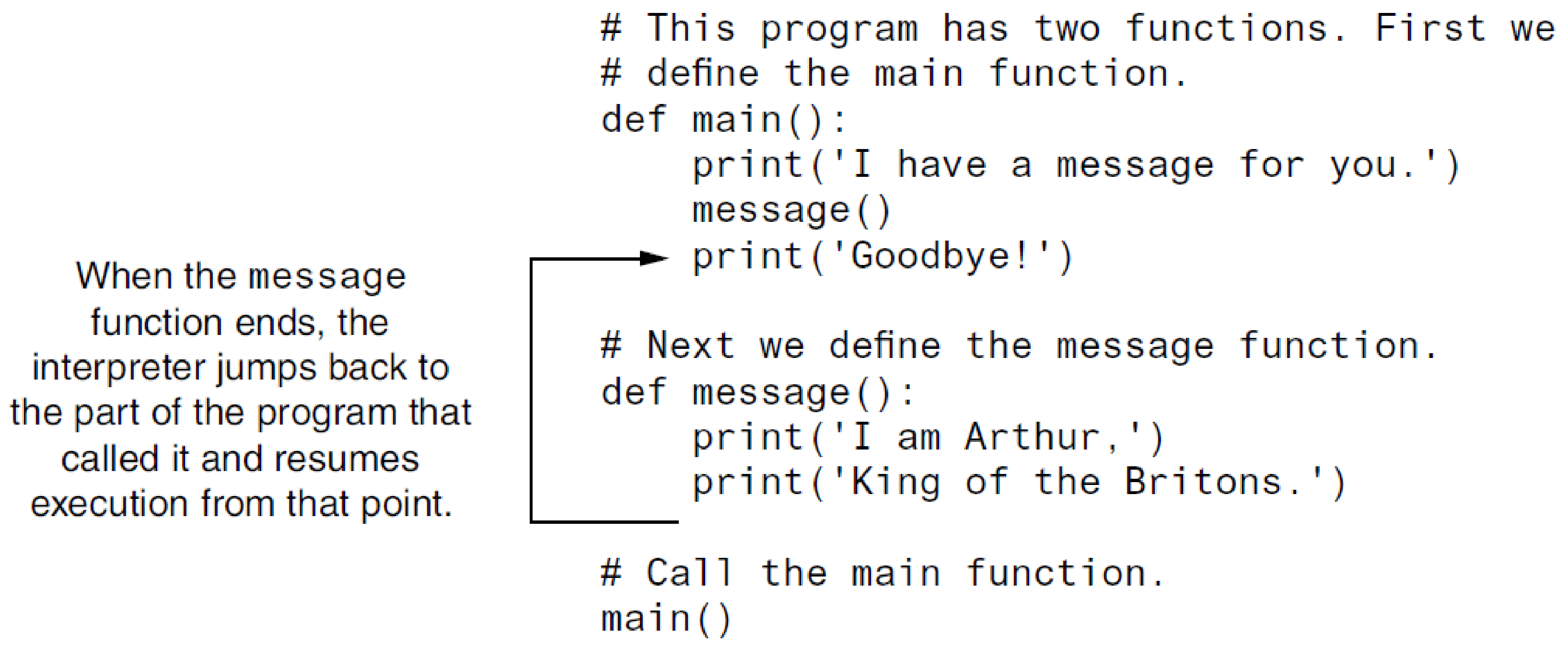
Aletheia University 資訊工程學系

The interpreter jumps to the main function and begins executing the statements in its block.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

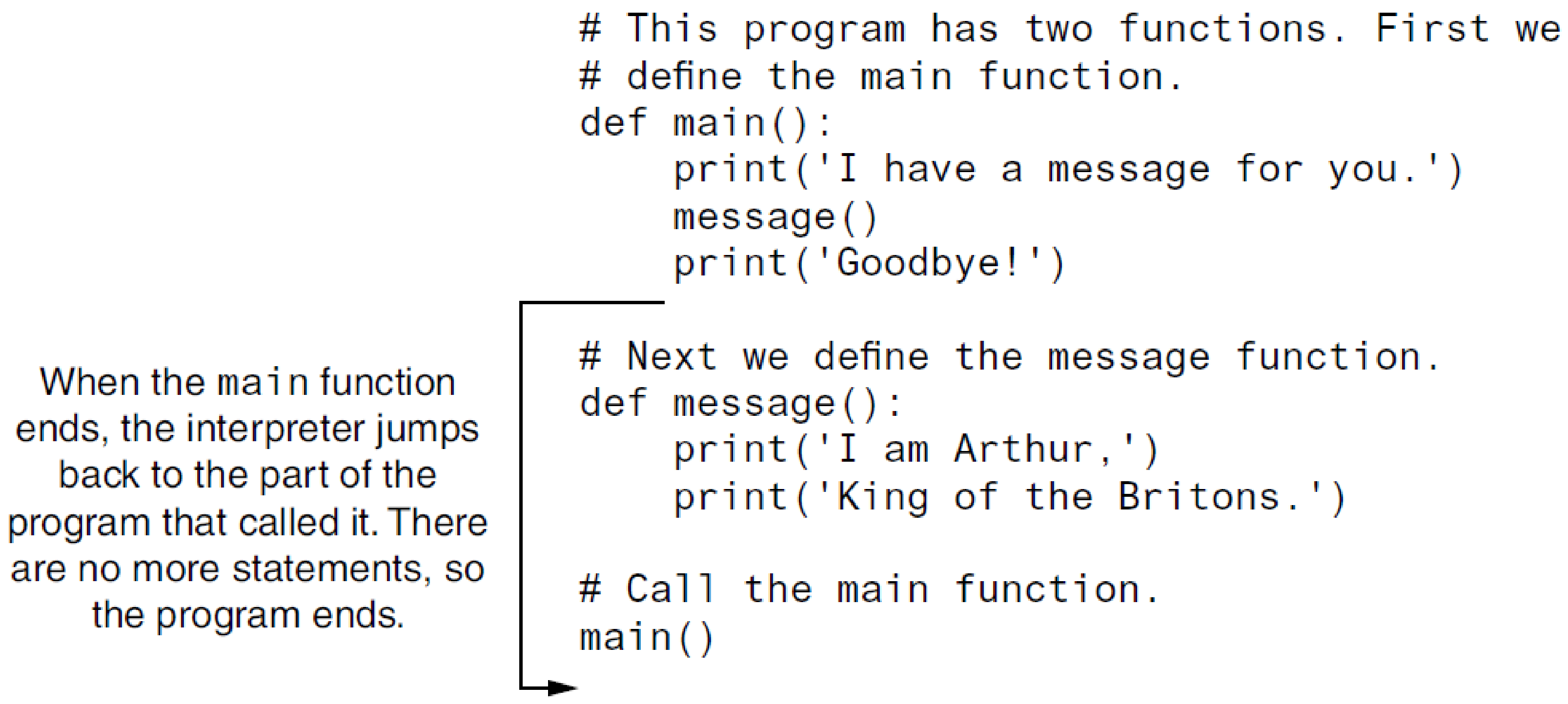
The interpreter jumps to the `message` function and begins executing the statements in its block.

```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```



When the message function ends, the interpreter jumps back to the part of the program that called it and resumes execution from that point.

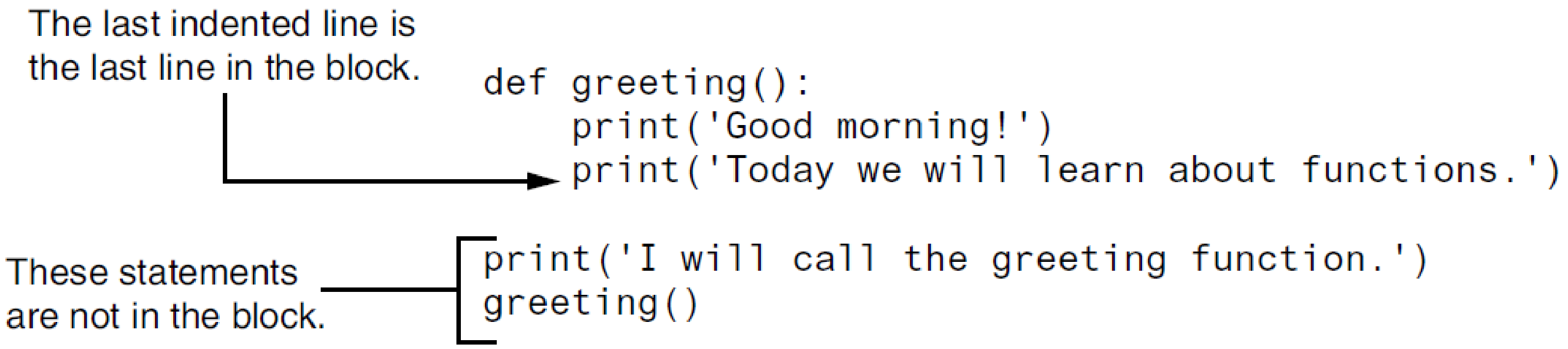
```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```



```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur,')
    print('King of the Britons.')

# Call the main function.
main()
```

When the main function ends, the interpreter jumps back to the part of the program that called it. There are no more statements, so the program ends.

Indentation in Python


The last indented line is the last line in the block.
These statements are not in the block.

```
def greeting():
    print('Good morning!')
    print('Today we will learn about functions.')

print('I will call the greeting function.')
greeting()
```

Local Variables

- A local variable is created **inside a function** and **cannot be accessed** by statements that are outside the function.
- Anytime you assign a value to a variable inside a function, you create a ***local variable***.
- Different functions can have **local variables with the same names** because the functions cannot see each other's local variables.

真理大學 Aletheia University 資訊工程學系

- An error will occur if a statement in one function tries to access a local variable that belongs to another function.

```
# Definition of the main function.
def main():
    get_name()
    print('Hello', name)      # This causes an error!

# Definition of the get_name function.
def get_name():
    name = input('Enter your name: ')

# Call the main function.
main()
```

- A variable’s *scope* is the part of a program in which the variable may be accessed.
- A local variable’s scope is the function in which the variable is created.
- These local variables will not overwrite one another since they exist in different “scopes”.

```
def newjersey():
  numbugs = 1000
  print (“NJ has”, numbugs, “bugs”)
def newyork():
  numbugs = 2000
  print (“NY has”, numbugs, “bugs”)

newjersey()
newyork()
```

真理大學 Aletheia University 資訊工程學系

- In addition, a local variable cannot be accessed by code that appears inside the function at a point **before the variable has been created**.

```
def bad_function():
    print('The value is', val) # This will cause an error!
    val = 99
```

Passing Arguments to Functions

- An ***argument*** is any piece of data that is passed into a function when the function is called.
- A **parameter is a variable** that receives an argument that is passed into a function.

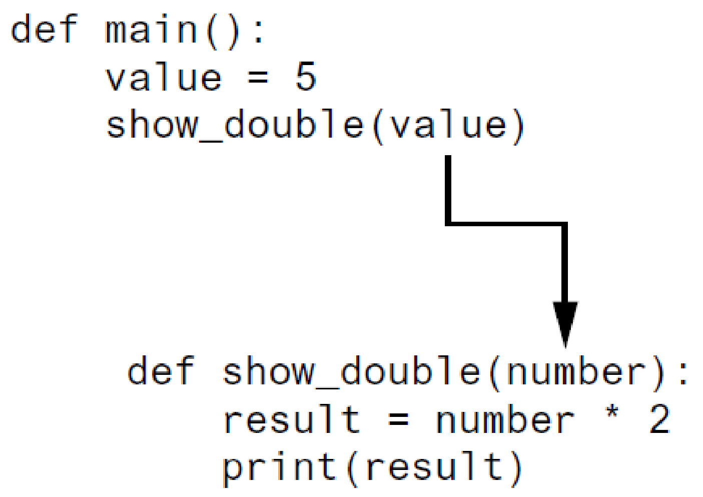
```
def show_double(number):
    result = number * 2
    print(result)
```

```python
# This program demonstrates an argument being
# passed to a function.

def main():
    value = 5
    show_double(value)

# The show_double function accepts an argument
# and displays double its value.
def show_double(number):
    result = number * 2
    print(result)

# Call the main function.
main()
```

Program Output

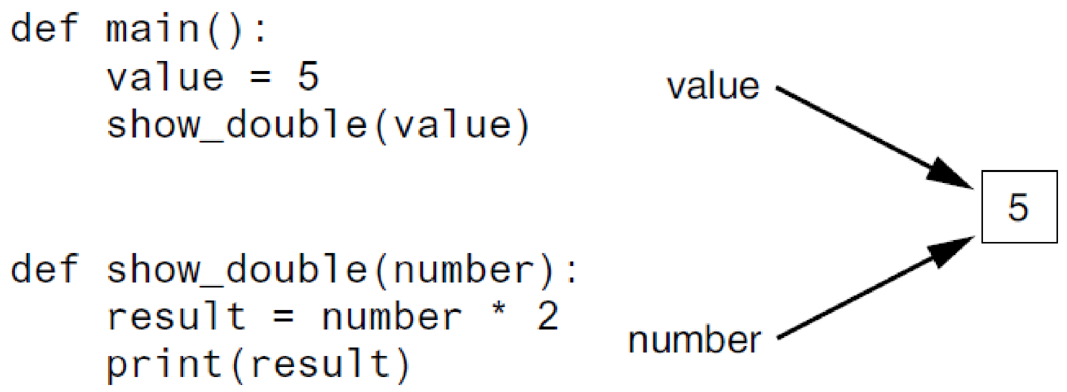
```
10
```

Aletheia University 資訊工程學系



```
def main():
    value = 5
    show_double(value)
```

```
def show_double(number):
    result = number * 2
    print(result)
```



```
def main():
    value = 5
    show_double(value)
```

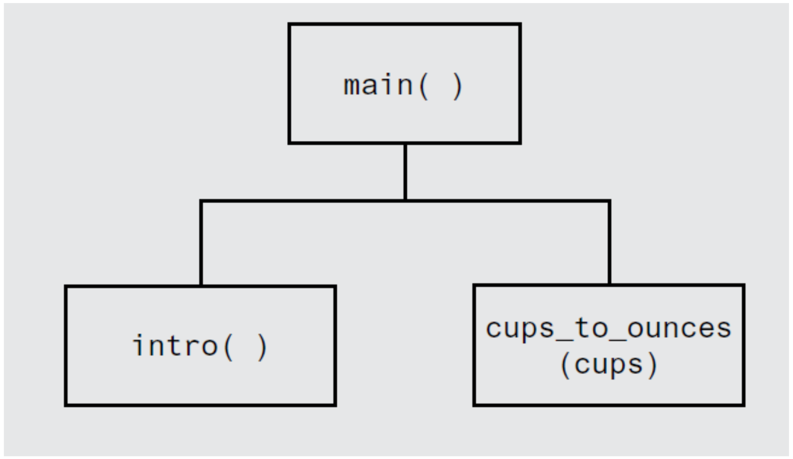
```
def show_double(number):
    result = number * 2
    print(result)
```

Parameter Variable Scope

- A variable is visible only to statements inside the variable's scope.
- A parameter variable's scope is the function in which the parameter is used.
- All of the statements inside the function can access the parameter variable, but no statement outside the function can access it.

- Your friend Michael runs a catering company. Some of the ingredients that his recipes require are measured in cups. When he goes to the grocery store to buy those ingredients, however, they are sold only by the fluid ounce. He has asked you to write a simple program that converts cups to fluid ounces.
- You design the following algorithm:
  1. *Display an introductory screen that explains what the program does.*
  2. *Get the number of cups.*
  3. *Convert the number of cups to fluid ounces and display the result.*
- This algorithm lists the top level of tasks that the program needs to perform and becomes the basis of the program's main function.

- The main function will call two other functions. Here are summaries of those functions:
  - intro. This function will display a message on the screen that explains what the program does.
  - cups_to_ounces. This function will accept the number of cups as an argument and calculate and display the equivalent number of fluid ounces.

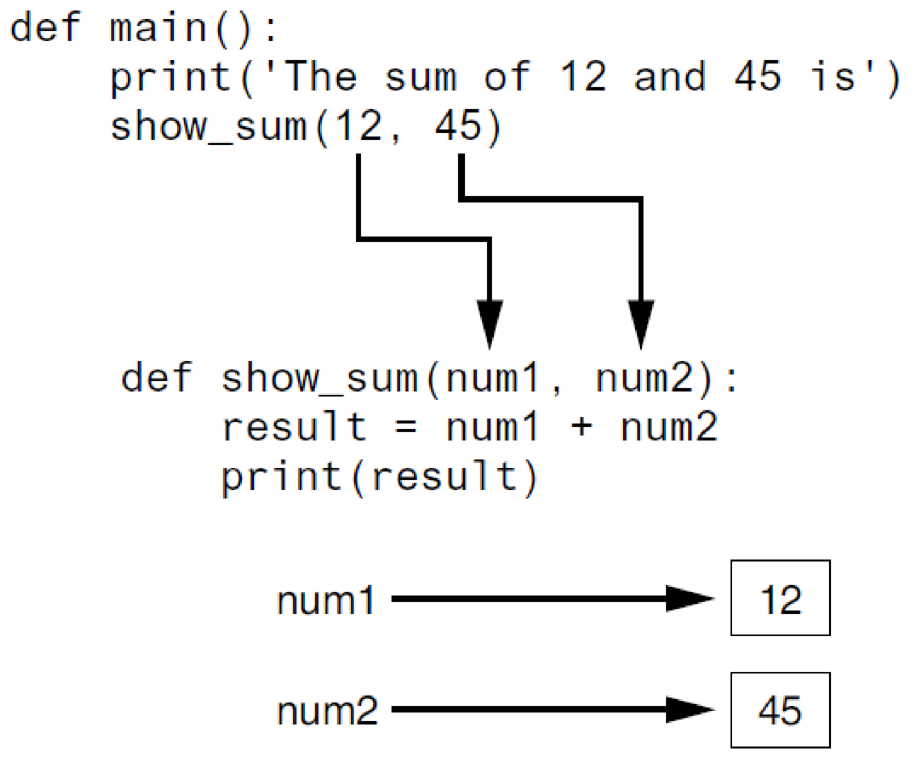


Program Output (with input shown in bold)

```
This program converts measurements
in cups to fluid ounces. For your
reference the formula is:
    1 cup = 8 fluid ounces
Enter the number of cups: 4 Enter
That converts to 32 ounces.
```

```
# This program converts cups to fluid ounces.

def main():
    # display the intro screen.
    intro()
    # Get the number of cups.
    cups_needed = int(input('Enter the number of cups: '))
    # Convert the cups to ounces.
    cups_to_ounces(cups_needed)

# The intro function displays an introductory screen.
def intro():
    print('This program converts measurements')
    print('in cups to fluid ounces. For your')
    print('reference the formula is:')
    print(' 1 cup = 8 fluid ounces')
    print()

# The cups_to_ounces function accepts a number of
# cups and displays the equivalent number of ounces.
def cups_to_ounces(cups):
    ounces = cups * 8
    print('That converts to', ounces, 'ounces.')

# Call the main function.
main()
```

Passing Multiple Arguments



```
def main():
    print('The sum of 12 and 45 is')
    show_sum(12, 45)
```

```
def show_sum(num1, num2):
    result = num1 + num2
    print(result)
```

```python
# This program demonstrates passing two string
# arguments to a function.

def main():
    first_name = input('Enter your first name: ')
    last_name = input('Enter your last name: ')
    print('Your name reversed is')
    reverse_name(first_name, last_name)

def reverse_name(first, last):
    print(last, first)

# Call the main function.
main()
```

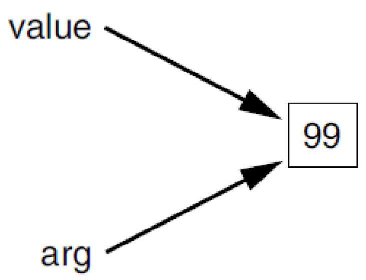
Program Output (with input shown in bold)

```
Enter your first name: Matt Enter
Enter your last name: Hoyle Enter
Your name reversed is
Hoyle Matt
```

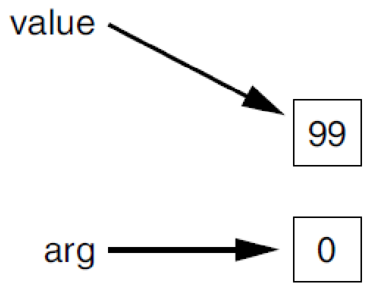
Making Changes to Parameters

- When an argument is passed to a function in Python, the function parameter variable will reference the argument's value.
- However, any changes that are made to the parameter variable will not affect the argument.

```
def main():
    value = 99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)

def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)
```



```
def main():
    value = 99
    print('The value is', value)
    change_me(value)
    print('Back in main the value is', value)

def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)
```



Argument Mechanics

- The form of argument passing that is used in Python, where a function **cannot change the value of an argument** that was passed to it, is commonly called ***pass by value***.
- We are essentially creating two copies of the data that is being passed – one that stays in the main program and one that is passed as an argument into our function.

Keyword Arguments

- Most programming languages match function arguments and parameters this way.
- Python language allows you to write an argument in the following format, to specify which parameter variable the argument should be passed to:

  parameter_name=value

- An argument that is written in accordance with this syntax is known as ***a keyword argument***.

```
# This program demonstrates keyword arguments.

def main():
    # Show the amount of simple interest, using 0.01 as
    # interest rate per period, 10 as the number of periods,
    # and $10,000 as the principal.
    show_interest(rate=0.01, periods=10, principal=10000.0)
    # The show_interest function displays the amount of
    # simple interest for a given principal, interest rate
    # per period, and number of periods

    def show_interest(principal, rate, periods):
        interest = principal * rate * periods
        print('The simple interest will be $',
              format(interest, ',.2f'),
              sep='')

    # Call the main function.
    main()
```

Global Variables

- When a variable is created by an assignment statement that is written outside all the functions in a program file, the variable is *global*.
- A global variable is accessible to all the functions in a program file.

```python
# Create a global variable.
number = 0

def main():
    global number
    number = int(input('Enter a number: '))
    show_number()

def show_number():
    print('The number you entered is', number)

# Call the main function.
main()
```

Program Output

```
Enter a number: 55 Enter
The number you entered is 55
```

```python
# This program demonstrates passing two strings as
# keyword arguments to a function.

def main():
    first_name = input('Enter your first name: ')
    last_name = input('Enter your last name: ')
    print('Your name reversed is')
    reverse_name(last=last_name, first=first_name)

def reverse_name(first, last):
    print(last, first)

# Call the main function.
main()
```

Program Output (with input shown in bold)

```
Enter your first name: Matt Enter
Enter your last name: Hoyle Enter
Your name reversed is
Hoyle Matt
```

Aletheia University 資訊工程學系

- If you want to be able to change a global variable insideof a function you must first tell Python that you wish to do this using the "global" keyword inside your function.
- Most programmers agree that you should restrict the use of global variables, or not use them at all. The reasons are as follows:
  - Global variables make debugging difficult.
  - Functions that use global variables are usually dependent on those variables.
  - Global variables make a program hard to understand.

Aletheia University
資訊工程學系

Value Returning Functions

- A value-returning function is a function that returns a value back to the part of the program that called it.
- A *value-returning function* is a special type of function. It is like a void function in the following ways.
  - It is a group of statements that perform a specific task.
  - When you want to execute the function, you call it.

```
def myfunction(arg1, arg2):
  statement
  statement
  …
  statement
  return expression


# call the function
returnvalue = myfunction(10, 50)
```

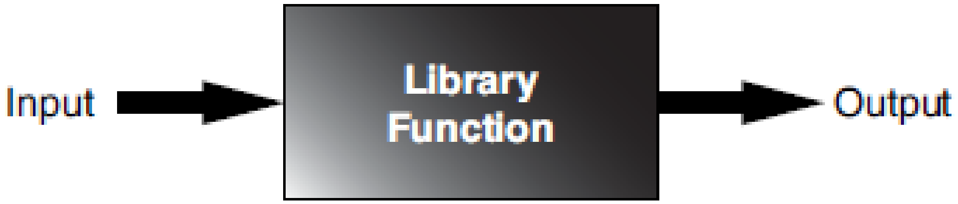
Standard Library Functions and the import Statement

- Python, as well as most programming languages, comes with a *standard library* of functions that have already been written for you.
- These functions, known as *library functions*, make a programmer's job easier because they perform many of the tasks that programmers commonly need to perform.
- Some of the functions that you have used are **print**, **input**, and **range**.

- If you want to use one of these built-in functions in a program, you simply call the function.
- Many of the functions in the standard library, however, are stored in files that are known as *modules*.
- In order to call a function that is stored in a module, you have to write an import statement at the top of your program.
- An import statement tells the interpreter the name of the module that contains the function.
- For example, one of the Python standard modules is named math.

```
import math
```

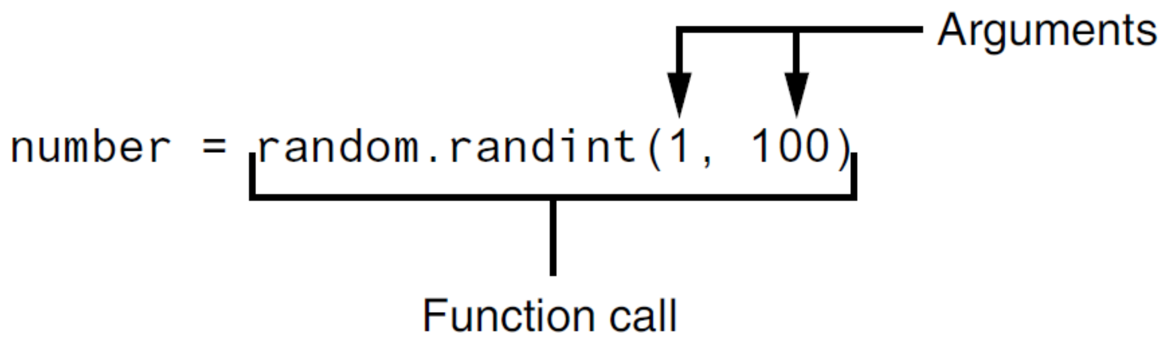
- This statement causes the interpreter to load the contents of the math module into memory and makes all the functions in the math module available to the program.
- Because you do not see the internal workings of library functions, many programmers think of them as *black boxes*.



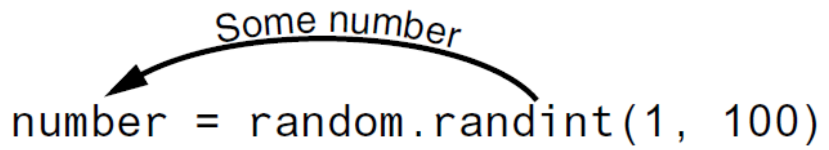
Generating Random Numbers

- Random numbers are useful for lots of different programming tasks. The following are just a few examples.
  - Random numbers are commonly used in games.
  - Random numbers are useful in simulation programs.
  - Random numbers are useful in statistical programs.
  - Random numbers are commonly used in computer security.
- To use any of these functions, you first need to write this import statement at the top of your program:

```
import random
```



```
number = random.randint(1, 100)
```



```
number = random.randint(1, 100)
```

A random number in the range of 1 through 100 will be assigned to the `number` variable.

```python
# This program displays a random number
# in the range of 1 through 10.
import random

def main():
    # Get a random number.
    number = random.randint(1, 10)
    # Display the number.
    print('The number is', number)

# Call the main function.
main()
```

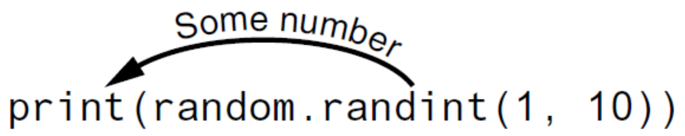
Program Output

```
The number is 7
```

```python
# This program displays five random
# numbers in the range of 1 through 100.
import random

def main():
    for count in range(5):
        # Get a random number.
        number = random.randint(1, 100)
        # Display the number.
        print(number)

# Call the main function.
main()
```

Program Output

```
89
7
16
41
12
```



```python
print(random.randint(1, 10))
```

A random number in the range of 1 through 10 will be displayed.

- Dr. Kimura teaches an introductory statistics class and has asked you to write a program that he can use in class to simulate the rolling of dice. The program should randomly generate two numbers in the range of 1 through 6 and display them. In your interview with Dr. Kimura, you learn that he would like to use the program to simulate several rolls of the dice, one after the other. Here is the pseudocode for the program:
- *While the user wants to roll the dice:*

```
    Display a random number in the range of 1 through 6
    Display another random number in the range of 1 through 6
    Ask the user if he or she wants to roll the dice again
```

- You will write a while loop that simulates one roll of the dice and then asks the user if another roll should be performed. As long as the user answers "y" for yes, the loop will repeat.

真理大學 Aletheia University 資訊工程學系

Program Output (with input shown in bold)

```
Rolling the dice . . .
Their values are:
3
1
Roll them again? (y = yes): y [Enter]
Rolling the dice . . .
Their values are:
1
1
Roll them again? (y = yes): y [Enter]
Rolling the dice . . .
Their values are:
5
6
Roll them again? (y = yes): y [Enter]
```

- Dr. Kimura was so happy with the dice rolling simulator that you wrote for him, he has asked you to write one more program. He would like a program that he can use to simulate ten coin tosses, one after the other. Each time the program simulates a coin toss, it should randomly display either "Heads" or "Tails".
- You decide that you can simulate the tossing of a coin by randomly generating a number in the range of 1 through 2. You will write an if statement that displays "Heads" if the random number is 1, or "Tails" otherwise. Here is the pseudocode:

```
Repeat 10 times:
    If a random number in the range of 1 through 2 equals 1 then:
        Display 'Heads'
    Else:
        Display 'Tails'
```

- Because the program should simulate 10 tosses of a coin you decide to use a for loop.

Program Output

```
Tails
Tails
Heads
Tails
Heads
Heads
Heads
Tails
Heads
Tails
```

The randrange, random, and uniform Functions

- The standard library's random module contains numerous functions for working with random numbers.
- In addition to the randint function, you might find the randrange, random, and uniform functions useful.
- For example, the following statement assigns a random number in the range of 0 through 9 to the number variable:

```
number = random.randrange(10)
```

- The following statement specifies both a starting value and an ending limit for the sequence:

```
number = random.randrange(5,10)
```

- The following statement specifies a starting value, an ending limit, and a step value:

```
number = random.randrange(0, 101, 10)
```

- The random function, however, returns a random floating-point number. When you call it, it returns a random floating point number in the range of **0.0 up to 1.0.**

```
number = random.random()
```

- The uniform function also returns a random floating-point number, but allows you to specify the range of values to select from.

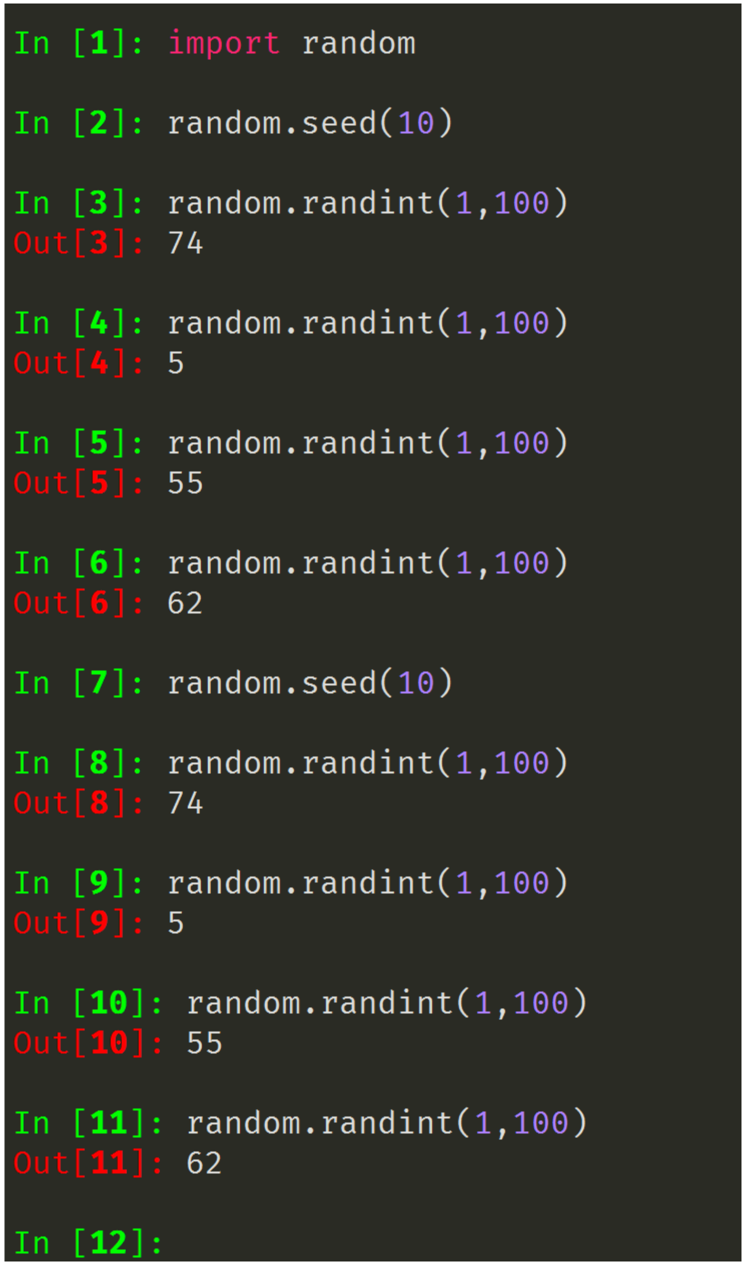
```
number = random.uniform(1.0, 10.0)
```

Random Number Seeds

- The numbers that are generated by the functions in the random module are not truly random.
- The formula that generates random numbers has to be initialized with a value known as a *seed value*.
- If the same seed value were always used, the random number functions would always generate the same series of pseudorandom numbers.
- You can call the random.seed function to specify a seed value. Here is an example:

```
random.seed(10)
```

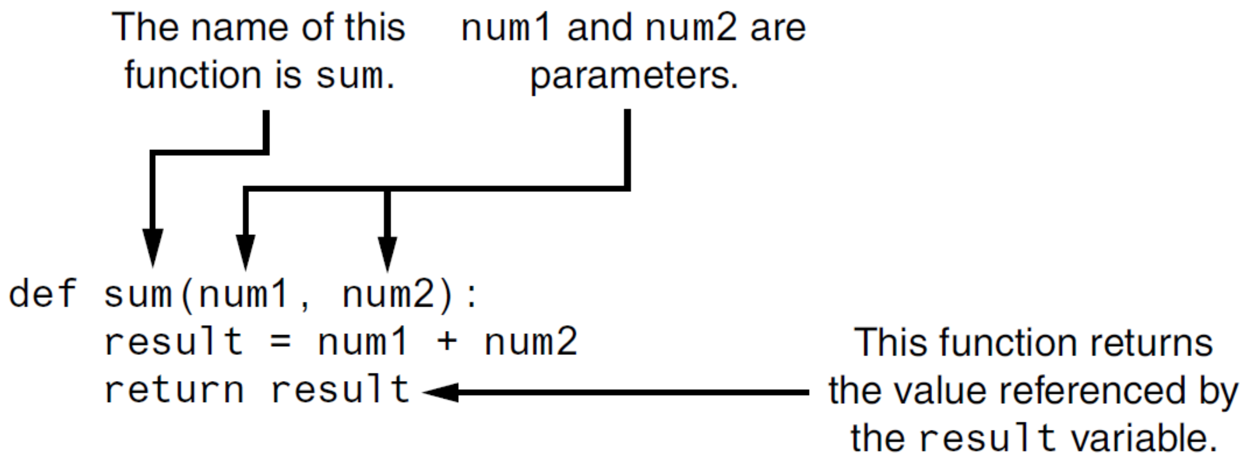
Aletheia University
資訊工程學系



```
In [1]: import random

In [2]: random.seed(10)

In [3]: random.randint(1,100)
Out[3]: 74

In [4]: random.randint(1,100)
Out[4]: 5

In [5]: random.randint(1,100)
Out[5]: 55

In [6]: random.randint(1,100)
Out[6]: 62

In [7]: random.seed(10)

In [8]: random.randint(1,100)
Out[8]: 74

In [9]: random.randint(1,100)
Out[9]: 5

In [10]: random.randint(1,100)
Out[10]: 55

In [11]: random.randint(1,100)
Out[11]: 62

In [12]:
```

Aletheia University 資訊工程學系

Writing Your Own Value-Returning Functions



The name of this function is sum.

num1 and num2 are parameters.

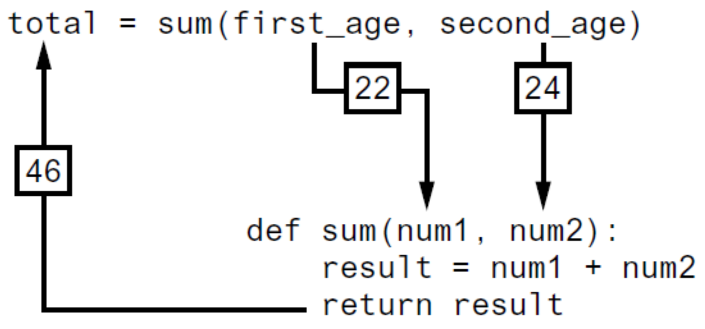
```
def sum(num1, num2):
    result = num1 + num2
    return result
```

This function returns the value referenced by the `result` variable.

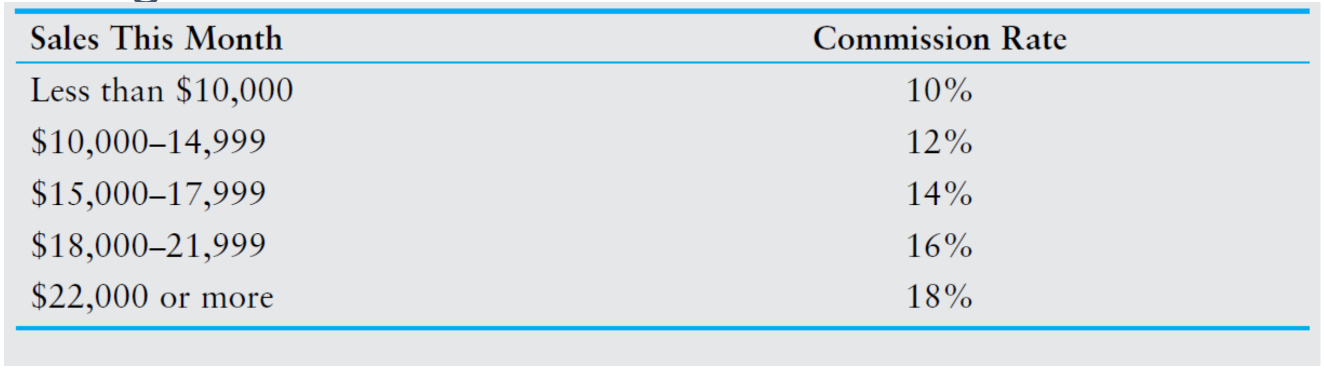
```python
# This program uses the return value of a function.

def main():
    # Get the user's age.
    first_age = int(input('Enter your age: '))

    # Get the user's best friend's age.
    second_age = int(input("Enter your best friend's age: "))

    # Get the sum of both ages.
    total = sum(first_age, second_age)

    # Display the total age.
    print('Together you are', total, 'years old.')

# The sum function accepts two numeric arguments and
# returns the sum of those arguments.
def sum(num1, num2):
    result = num1 + num2
    return result

# Call the main function.
main()
```

Program Output (with input shown in bold)

```
Enter your age: 22 Enter
Enter your best friend's age: 24 Enter
Together you are 46 years old.
```



- Hal owns a business named Make Your Own Music, which sells guitars, drums, banjos, synthesizers, and many other musical instruments. Hal's sales staff works strictly on commission. At the end of the month, each salesperson's commission is calculated according to Table.



Sales This Month	Commission Rate
Less than $10,000	10%
$10,000–14,999	12%
$15,000–17,999	14%
$18,000–21,999	16%
$22,000 or more	18%

- Because the staff gets paid once per month, Hal allows each employee to take up to $2,000 per month in advance. When sales commissions are calculated, the amount of each employee's advanced pay is subtracted from the commission. If any salesperson's commissions are less than the amount of their advance, they must reimburse Hal for the difference. To calculate a salesperson's monthly pay, Hal uses the following formula:

$$\textit{pay} = \textit{sales} * \textit{commission rate} - \textit{advanced pay}$$

- Hal has asked you to write a program that makes this calculation for him. The following general algorithm outlines the steps the program must take.

  1. *Get the salesperson's monthly sales.*
  2. *Get the amount of advanced pay.*
  3. *Use the amount of monthly sales to determine the commission rate.*
  4. *Calculate the salesperson's pay using the formula previously shown. If the amount is negative, indicate that the salesperson must reimburse the company.*

Program Output (with input shown in bold)

```
Enter the monthly sales: 14650.00 Enter
Enter the amount of advanced pay, or
enter 0 if no advanced pay was given.
Advanced pay: 1000.00 Enter
The pay is $758.00
```

Program Output (with input shown in bold)

```
Enter the monthly sales: 9000.00 Enter
Enter the amount of advanced pay, or
enter 0 if no advanced pay was given.
Advanced pay: 0 Enter
The pay is $900.00
```

Program Output (with input shown in bold)

```
Enter the monthly sales: 12000.00 Enter
Enter the amount of advanced pay, or
enter 0 if no advanced pay was given.
Advanced pay: 2000.00 Enter
The pay is $-560.00
The salesperson must reimburse
the company.
```

Returning Strings

```
def get_name():
    # Get the user's name.
    name = input('Enter your name: ')
    # Return the name.
    return name
```

Returning Boolean Values

- Python allows you to write *Boolean functions*, which return either True or False.

```
def is_even(number):
    # Determine whether number is even. If it is,
    # set status to true. Otherwise, set status
    # to false.
    if (number % 2) == 0:
        status = True
    else:
        status = False
    # Return the value of the status variable.
    return status
```

Aletheia University
資訊工程學系

Returning Multiple Values

- In Python, however, you are not limited to returning only one value. You can specify multiple expressions separated by commas after the return statement.

return *expression1*, *expression2*, *etc.*

```
def get_name():
    # Get the user's first and last names.
    first = input('Enter your first name: ')
    last = input('Enter your last name: ')

    # Return both names.
    return first, last
```

```
first_name, last_name = get_name()
```

The math Module

- The Python standard library's math module contains numerous functions that can be used in mathematical calculations.

```
# This program demonstrates the sqrt function.
import math

def main():
    # Get a number.
    number = float(input('Enter a number: '))

    # Get the square root of the number.
    square_root = math.sqrt(number)

    # Display the square root.
    print('The square root of', number, '0 is', square_root)

# Call the main function.
main()
```

Program Output (with input shown in bold)

```
Enter a number: 25 Enter
The square root of 25.0 is 5.0
```

真理大學 Aletheia University 資訊工程學系

`math` Module Function	Description
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x, y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

The math.pi and math.e Values

- The math module also defines two variables, pi and e, which are assigned mathematical values for *pi* and e.

```
area = math.pi * radius**2
```

Aletheia University
資訊工程學系

Storing Functions in Modules

- A module is a file that contains Python code.
- Large programs are easier to debug and maintain when they are divided into modules.
- When you break a program into modules, each module should contain functions that perform related tasks.
- If you have written a set of functions that are needed in several different programs, you can place those functions in a module.
- Then, you can import the module in each program that needs to call one of the functions.

- Let’s look at a simple example. Suppose your instructor has asked you to write a program that calculates the following:
  - The area of a circle
  - The circumference of a circle
  - The area of a rectangle
  - The perimeter of a rectangle

circle.py

```python
# The circle module has functions that perform
# calculations related to circles.
import math

# The area function accepts a circle's radius as an
# argument and returns the area of the circle.
def area(radius):
    return math.pi * radius**2

# The circumference function accepts a circle's
# radius and returns the circle's circumference.
def circumference(radius):
    return 2 * math.pi * radius
```

Aletheia University 資訊工程學系

rectangle.py

```python
# The rectangle module has functions that perform
# calculations related to rectangles.

# The area function accepts a rectangle's width and
# length as arguments and returns the rectangle's area.
def area(width, length):
    return width * length

# The perimeter function accepts a rectangle's width
# and length as arguments and returns the rectangle's
# perimeter.
def perimeter(width, length):
    return 2 * (width + length)
```

- we should mention the following things about module names:
  - A module's file name should end in .py. If the module's file name does not end in .py, you will not be able to import it into other programs.
  - A module's name cannot be the same as a Python key word. An error would occur, for example, if you named a module for.

```python
# This program allows the user to choose various
# geometry calculations from a menu. This program
# imports the circle and rectangle modules.
import circle
import rectangle

# Constants for the menu choices
AREA_CIRCLE_CHOICE = 1
CIRCUMFERENCE_CHOICE = 2
AREA_RECTANGLE_CHOICE = 3
PERIMETER_RECTANGLE_CHOICE = 4
QUIT_CHOICE = 5

# The main function.
def main():
    # The choice variable controls the loop
    # and holds the user's menu choice.
    choice = 0

    while choice != QUIT_CHOICE:
        # display the menu.
        display_menu()

        # Get the user's choice.
        choice = int(input('Enter your choice: '))

        # Perform the selected action.
        if choice == AREA_CIRCLE_CHOICE:
            radius = float(input("Enter the circle's radius: "))
            print('The area is', circle.area(radius))
        elif choice == CIRCUMFERENCE_CHOICE:
            radius = float(input("Enter the circle's radius: "))
            print('The circumference is',
                  circle.circumference(radius))
        elif choice == AREA_RECTANGLE_CHOICE:
            width = float(input("Enter the rectangle's width: "))
            length = float(input("Enter the rectangle's length: "))
            print('The area is', rectangle.area(width, length))
        elif choice == PERIMETER_RECTANGLE_CHOICE:
            width = float(input("Enter the rectangle's width: "))
            length = float(input("Enter the rectangle's length: "))
            print('The perimeter is',
                  rectangle.perimeter(width, length))
        elif choice == QUIT_CHOICE:
            print('Exiting the program. . .')
        else:
            print('Error: invalid selection.')

# The display_menu function displays a menu.
def display_menu():
    print('        MENU')
    print('1) Area of a circle')
    print('2) Circumference of a circle')
    print('3) Area of a rectangle')
    print('4) Perimeter of a rectangle')
    print('5) Quit')

# Call the main function.
main()
```

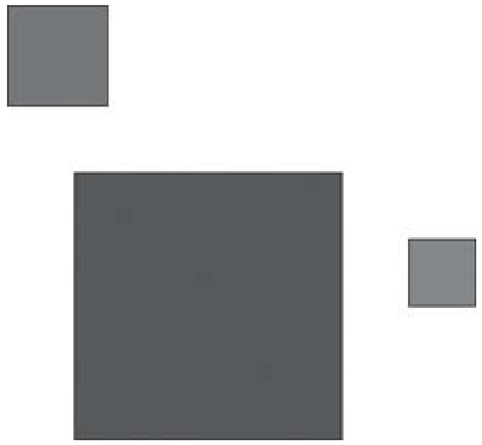
Program Output (with input shown in bold)

```
      MENU
1) Area of a circle
2) Circumference of a circle
3) Area of a rectangle
4) Perimeter of a rectangle
5) Quit
Enter your choice: 1 Enter
Enter the circle's radius: 10
The area is 314.159265359
      MENU
1) Area of a circle
2) Circumference of a circle
3) Area of a rectangle
4) Perimeter of a rectangle
5) Quit
Enter your choice: 2 Enter
Enter the circle's radius: 10
The circumference is 62.8318530718
      MENU
1) Area of a circle
2) Circumference of a circle
3) Area of a rectangle
4) Perimeter of a rectangle
5) Quit
```

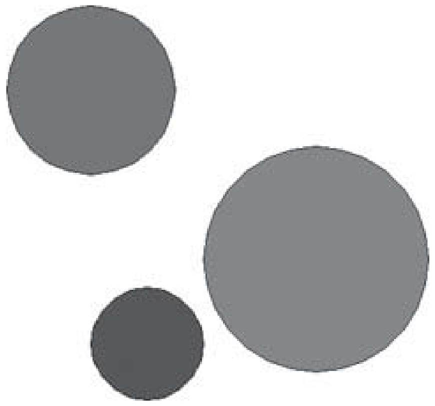
Turtle Graphics: Modularizing Code with Functions

- Commonly needed turtle graphics operations can be stored in functions and then called whenever needed.

```python
import turtle

def main():
    turtle.hideturtle()
    square(100, 0, 50, 'red')
    square(-150, -100, 200, 'blue')
    square(-200, 150, 75, 'green')

# The square function draws a square. The x and y parameters
# are the coordinates of the lower-left corner. The width
# parameter is the width of each side. The color parameter
# is the fill color, as a string.

def square(x, y, width, color):
    turtle.penup()              # Raise the pen
    turtle.goto(x, y)           # Move to the specified location
    turtle.fillcolor(color)     # Set the fill color
    turtle.pendown()            # Lower the pen
    turtle.begin_fill()         # Start filling
    for count in range(4):      # Draw a square
        turtle.forward(width)
        turtle.left(90)
    turtle.end_fill()           # End filling

# Call the main function.
main()
```



```python
import turtle

def main():
    turtle.hideturtle()
    circle(0, 0, 100, 'red')
    circle(-150, -75, 50, 'blue')
    circle(-200, 150, 75, 'green')

# The circle function draws a circle. The x and y parameters
# are the coordinates of the center point. The radius
# parameter is the circle's radius. The color parameter
# is the fill color, as a string.

def circle(x, y, radius, color):
    turtle.penup()              # Raise the pen
    turtle.goto(x, y - radius)  # Position the turtle
    turtle.fillcolor(color)     # Set the fill color
    turtle.pendown()            # Lower the pen
    turtle.begin_fill()         # Start filling
    turtle.circle(radius)       # Draw a circle
    turtle.end_fill()           # End filling

# Call the main function.
main()
```



```python
import turtle

# Named constants for the triangle's points
TOP_X = 0
TOP_Y = 100
BASE_LEFT_X = -100
BASE_LEFT_Y = -100
BASE_RIGHT_X = 100
BASE_RIGHT_Y = -100

def main():
    turtle.hideturtle()
    line(TOP_X, TOP_Y, BASE_LEFT_X, BASE_LEFT_Y, 'red')
    line(TOP_X, TOP_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'blue')
    line(BASE_LEFT_X, BASE_LEFT_Y, BASE_RIGHT_X, BASE_RIGHT_Y, 'green')

# The line function draws a line from (startX, startY)
# to (endX, endY). The color parameter is the line's color.

def line(startX, startY, endX, endY, color):
    turtle.penup()                  # Raise the pen
    turtle.goto(startX, startY)     # Move to the starting point
    turtle.pendown()                # Lower the pen
    turtle.pencolor(color)          # Set the pen color
    turtle.goto(endX, endY)         # Draw a square

# Call the main function.
main()
```