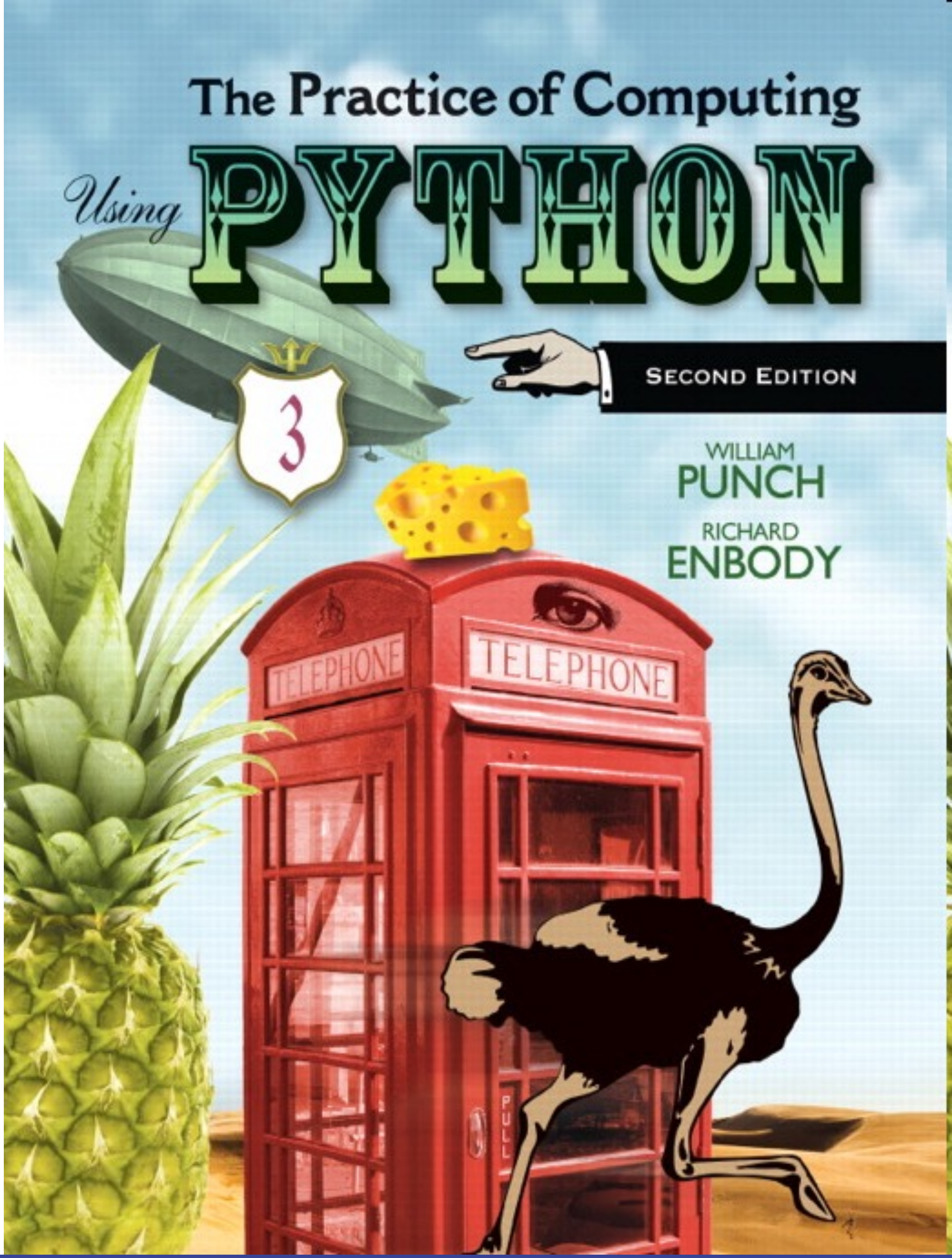
chapter 5

# Functions -- QuickStart



## What is a function?

## Functions

- From Mathematics we know that functions perform some operation and return one value.
- They "encapsulate" the performance of some particular operation, so it can be used by others (for example, the `sqrt()` function)

# Why have them?

- Support divide-and-conquer strategy
- Abstraction of an operation
- Reuse. Once written, use again
- Sharing. If tested, others can use
- Security. Well tested, then secure for reuse
- Simplify code. More readable.

# Mathematical Notation

- Consider a function which converts temperatures in Celsius to temperatures in Fahrenheit.
  - Formula: F = C * 1.8 + 32.0
  - Functional notation:

    F ~ celsius_to_Fahrenheit(C) where

    celsius_to_Fahrenheit(C) = C * 1.8 + 32.0

## Python Invocation

- Math: F = celsius_to_Fahrenheit(C)
- Python, the invocation is much the same

```
F = celsius_to_Fahrenheit(cel_float)
```

Terminology: `cel_float` is the ***argument***

## Function defintion

- Math: g(C) = C*1.8 + 32.0
- Python

```
def celsius_to_Fahrenheit(param_float):
    return param_float * 1.8 + 32.0
```

- Terminology: `cel_float` is the ***parameter***

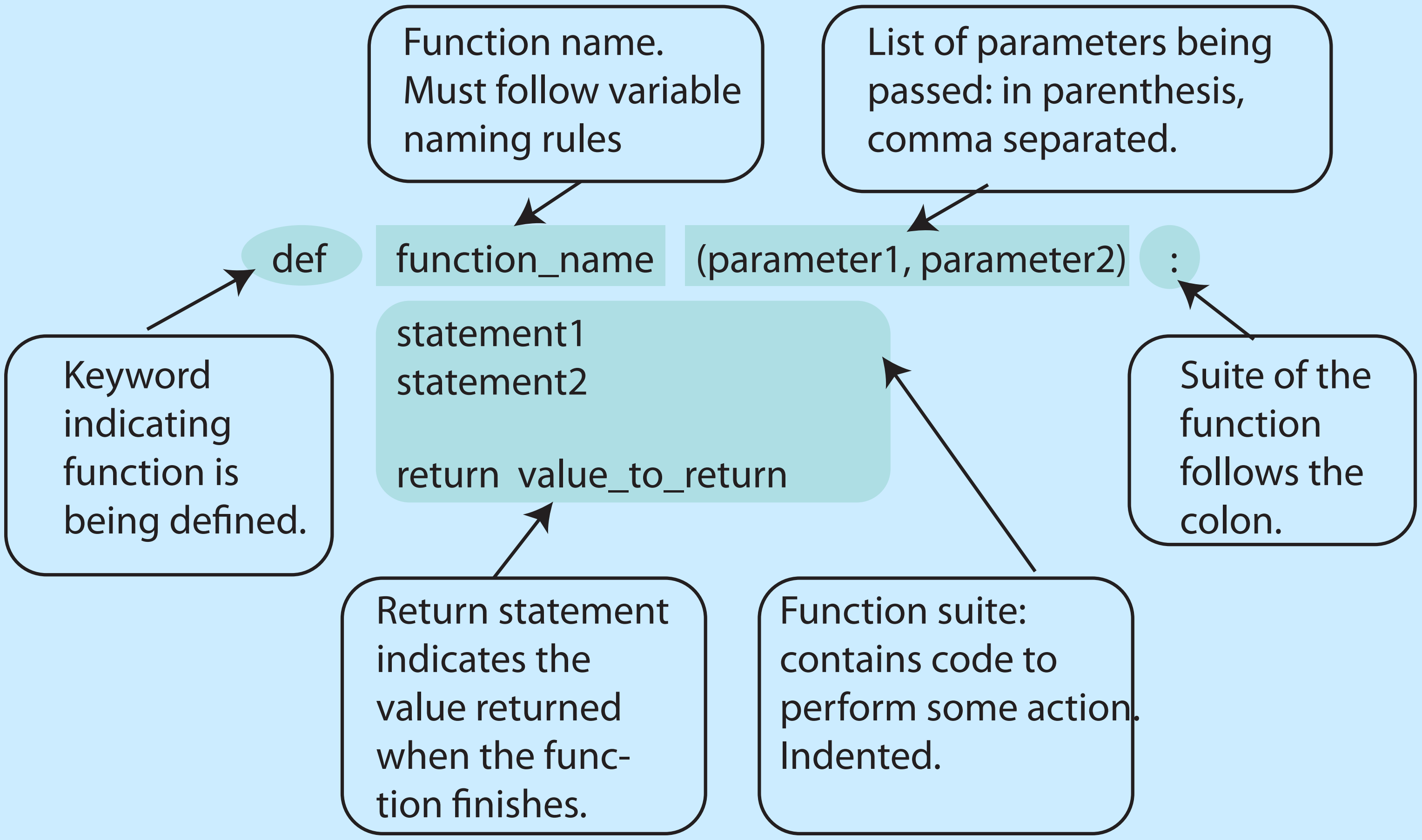

Function name.
Must follow variable naming rules
List of parameters being passed: in parenthesis, comma separated.
def
function_name
(parameter1, parameter2)
:
statement1
statement2
return value_to_return
Keyword indicating function is being defined.
Suite of the function follows the colon.
Return statement indicates the value returned when the func- tion finishes.
Function suite: contains code to perform some action. Indented.

Figure 5.1 Function Parts

## return statement

- The `return` statement indicates the value that is returned by the function
- The statement is optional (the function can return nothing). If no `return`, function is often called a procedure.

## Code Listing 5.1
## Temp convert

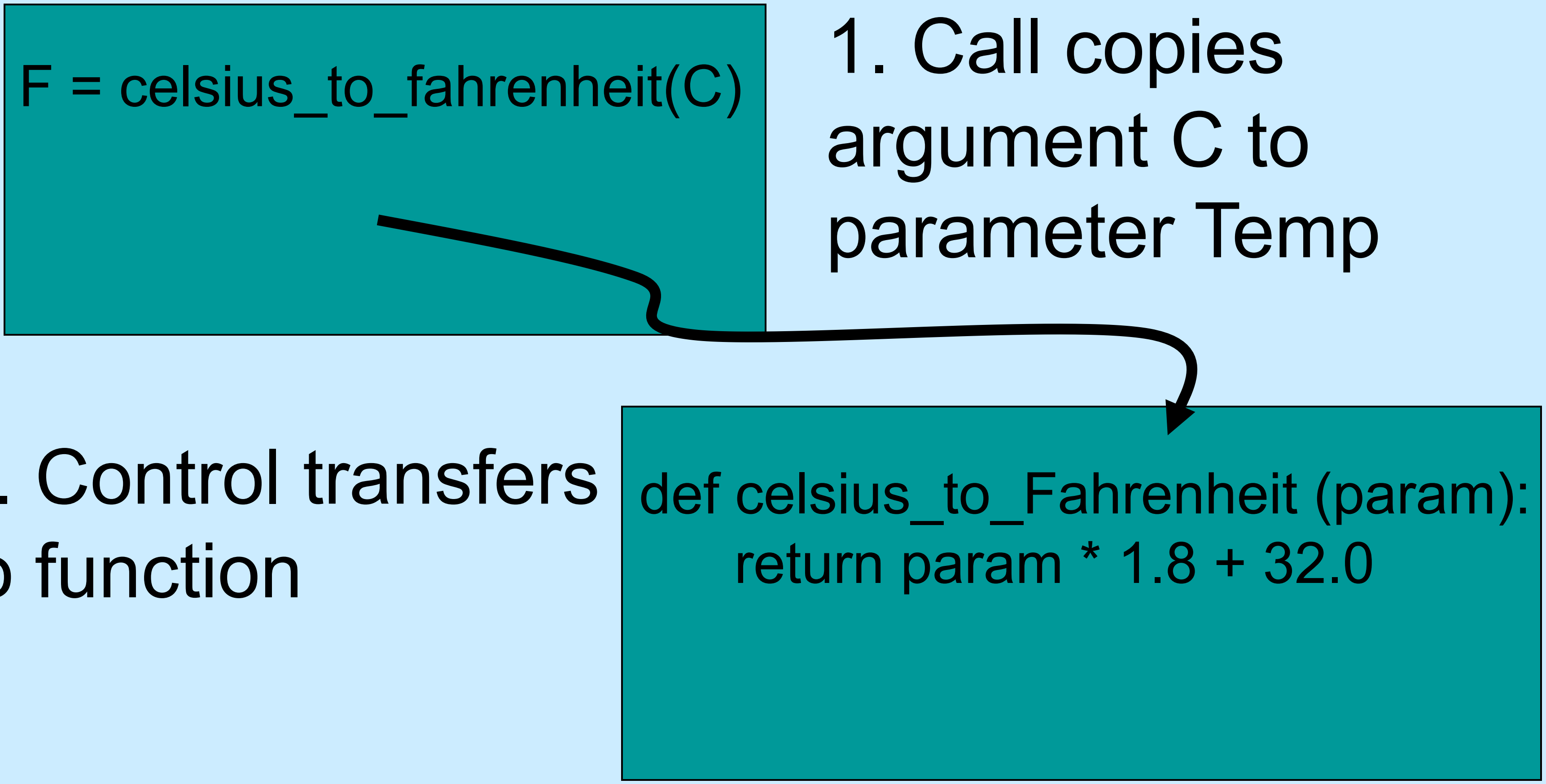
```python
# Temperature conversion

def celsius_to_fahrenheit(celsius_float):
    """ Convert Celsius to Fahrenheit."""
    return celsius_float * 1.8 + 32
```

## Triple quoted string in function

- A triple quoted string just after the def is called a ***docstring***
- docstring is documentation of the function's purpose, to be used by other tools to tell the user what the function is used for. More on that later

# Operation



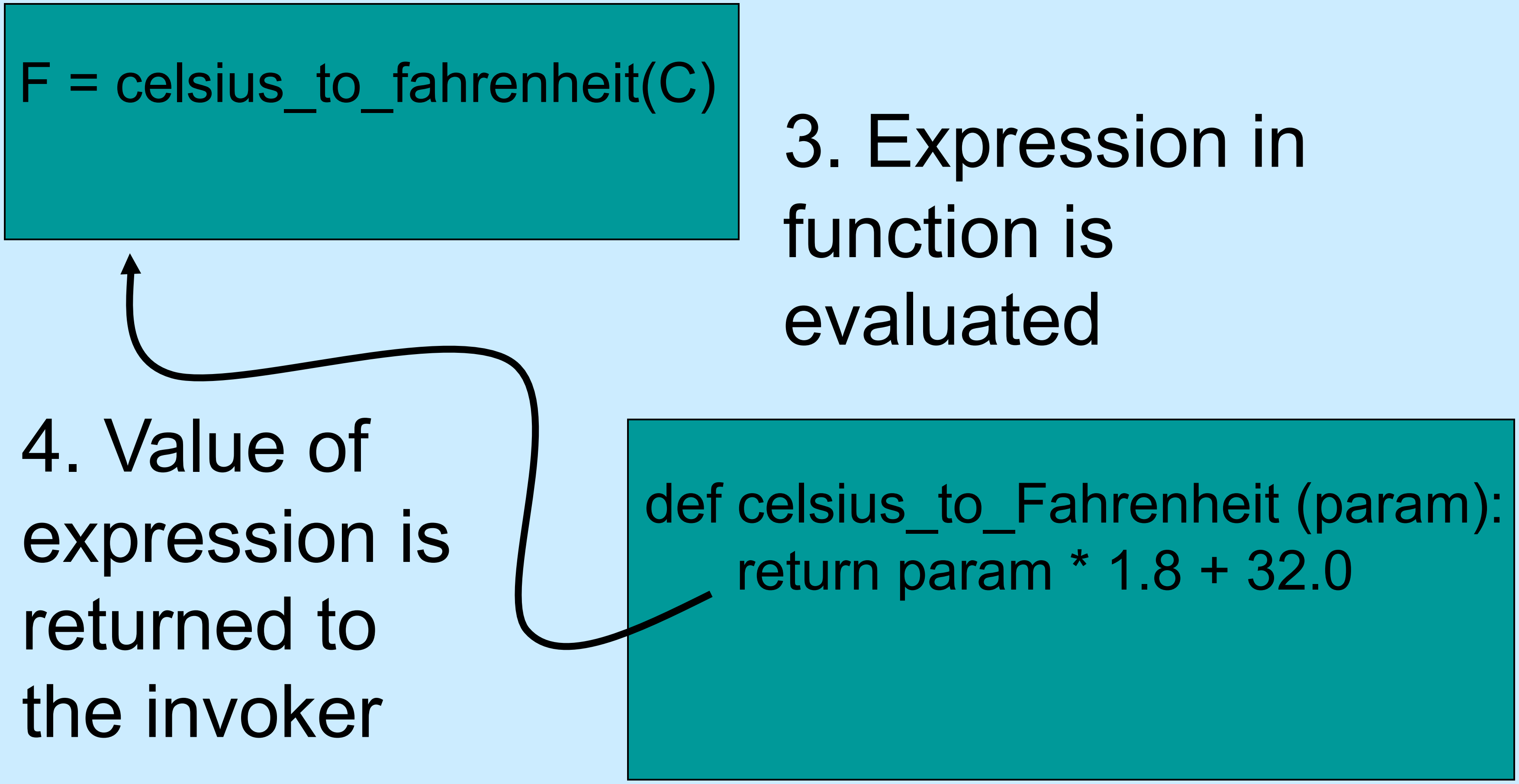
```
F = celsius_to_fahrenheit(C)
```

1. Call copies argument C to parameter Temp

```
def celsius_to_Fahrenheit (param):
    return param * 1.8 + 32.0
```

2. Control transfers to function

# Operation (con't)



```
F = celsius_to_fahrenheit(C)
```

3. Expression in function is evaluated

4. Value of expression is returned to the invoker

```
def celsius_to_Fahrenheit (param):
    return param * 1.8 + 32.0
```

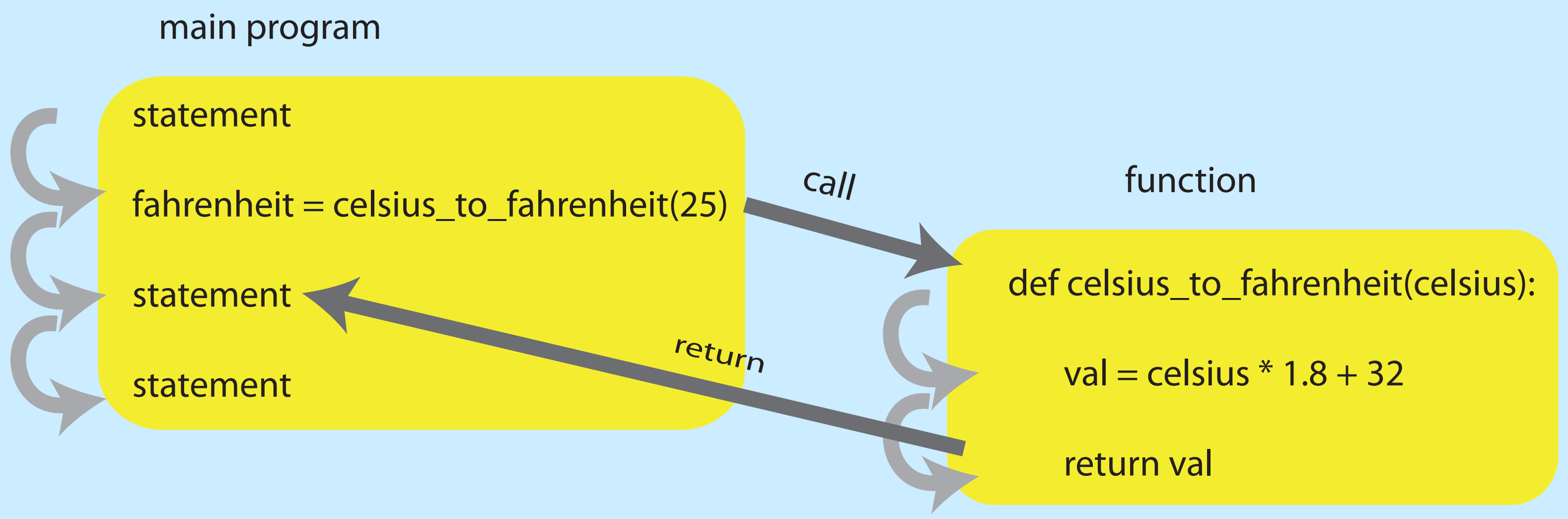


Figure 5.1 Function flow of control

## Code Listing 5.2
## Full Temp Program

```python
# Conversion program

def celsius_to_fahrenheit(celsius_float):
    """ Convert Celsius to Fahrenheit."""
    return celsius_float * 1.8 + 32

# main part of the program
print("Convert Celsius to Fahrenheit.")
celsius_float = float(input("Enter a Celsius temp: "))
# call the conversion function
fahrenheit_float = celsius_to_fahrenheit(celsius_float)
# print the returned value
print(celsius_float," converts to ",fahrenheit_float," Fahrenheit")
```

## Code Listing 5.3
## digit extraction

```python
def get_digit(number, position):
    '''return digit at position in number, counting from right'''
    return number//(10**position)%10
```

## Area of a Triangle

The next few functions can be used together to find the area of a triangle.

Note how we decompose the problem and then re-assemble the overall solution using the functions created

## Code Listing 5.4
## Input

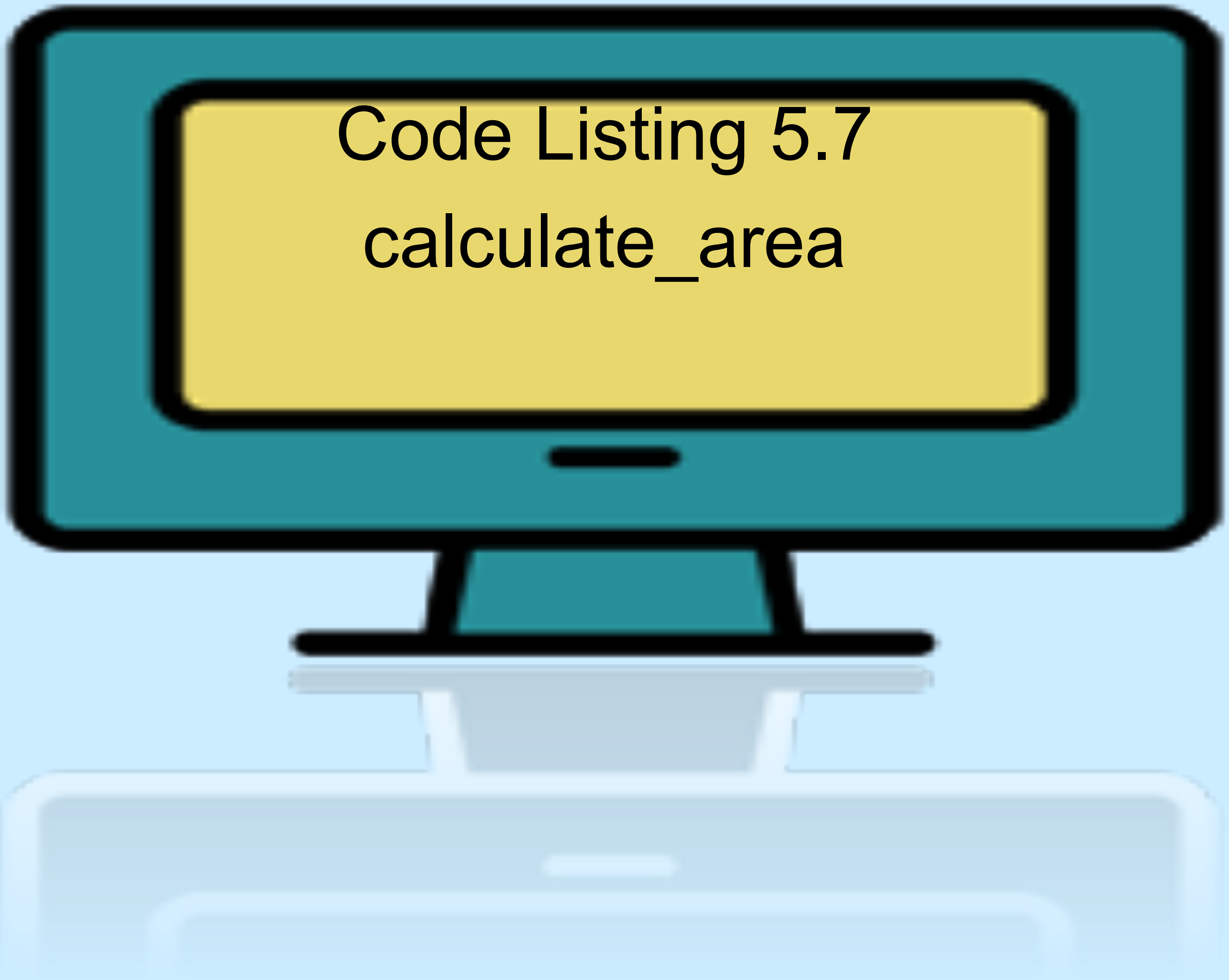
```
def get_vertex():
    x = float(input("    Please enter x: "))
    y = float(input("    Please enter y: "))
    return x,y
```

## Code Listing 5.5
## get_triangle

```
def get_triangle():
    print("First vertex")
    x1,y1 = get_vertex()
    print("Second vertex")
    x2,y2 = get_vertex()
    print("Third vertex")
    x3,y3 = get_vertex()
    return x1, y1, x2, y2, x3, y3
```

## Code Listing 5.6
## side_length

```
def side_length(x1,y1,x2,y2):
    ''' return length of a side (Euclidean distance) '''
    return math.sqrt((x1-x2)**2 + (y1-y2)**2)
```



## Code Listing 5.7
## calculate_area

```
def calculate_area(x1,y1,x2,y2,x3,y3):
    ''' return area using Heron's formula '''
    a = side_length(x1,y1,x2,y2)
    b = side_length(x2,y2,x3,y3)
    c = side_length(x3,y3,x1,y1)
    s = (1/2)*(a + b + c)
    return math.sqrt(s*(s-a)*(s-b)*(s-c))
```

## Code Listing 5.8
## Full Triangle Program

```
import math

def get_vertex():
    x = float(input("    Please enter x: "))
    y = float(input("    Please enter y: "))
    return x,y

def get_triangle():
    print("First vertex")
    x1,y1 = get_vertex()
    print("Second vertex")
    x2,y2 = get_vertex()
    print("Third vertex")
    x3,y3 = get_vertex()
    return x1, y1, x2, y2, x3, y3

def side_length(x1,y1,x2,y2):
    '''return length of a side (Euclidean distance)'''
    return math.sqrt((x1-x2)**2 + (y1-y2)**2)

def calculate_area(x1,y1,x2,y2,x3,y3):
    '''return area using Heron's forumula'''
    a = side_length(x1,y1,x2,y2)
    b = side_length(x2,y2,x3,y3)
    c = side_length(x3,y3,x1,y1)
    s = (1/2)*(a + b + c)
    return math.sqrt(s*(s-a)*(s-b)*(s-c))

x1, y1, x2, y2, x3, y3 = get_triangle()
area = calculate_area(x1,y1,x2,y2,x3,y3)
print("Area is",area)
```

## Did functions help?

- Made our problem solving easier (solved smaller problems as functions)
- main program very readable (details hid in the functions)

# How to write a function

- ***Does one thing***. If it does too many things, it should be broken down into multiple functions (refactored)
- ***Readable.*** How often should we say this? If you write it, it should be readable
- ***Reusable***. If it does one thing well, then when a similar situation (in another program) occurs, use it there as well.

# More on functions

- ***Complete***. A function should check for all the cases where it might be invoked. Check for potential errors.
- ***Not too long***. Kind of synonymous with do one thing. Use it as a measure of doing too much.

## Rule 8

A function should do one thing

## Procedures

- Functions that have no return statements are often called *procedures*.
- Procedures are used to perform some duty (print output, store a file, etc.)
- Remember, return is not required.

## Multiple returns in a function

- A function can have multiple `return` statements.
- Remember, the first `return` statement executed ends the function.
- Multiple returns can be confusing to the reader and should be used judiciously.

# Reminder, rules so far

1. Think before you program!
2. A program is a human-readable essay on problem solving that also happens to execute on a computer.
3. The best way to imporve your programming and problem solving skills is to practice!
4. A foolish consistency is the hobgoblin of little minds
5. Test your code, often and thoroughly
6. If it was hard to write, it is probably hard to read. Add a comment.
7. All input is evil, unless proven otherwise.
8. A function should do one thing.