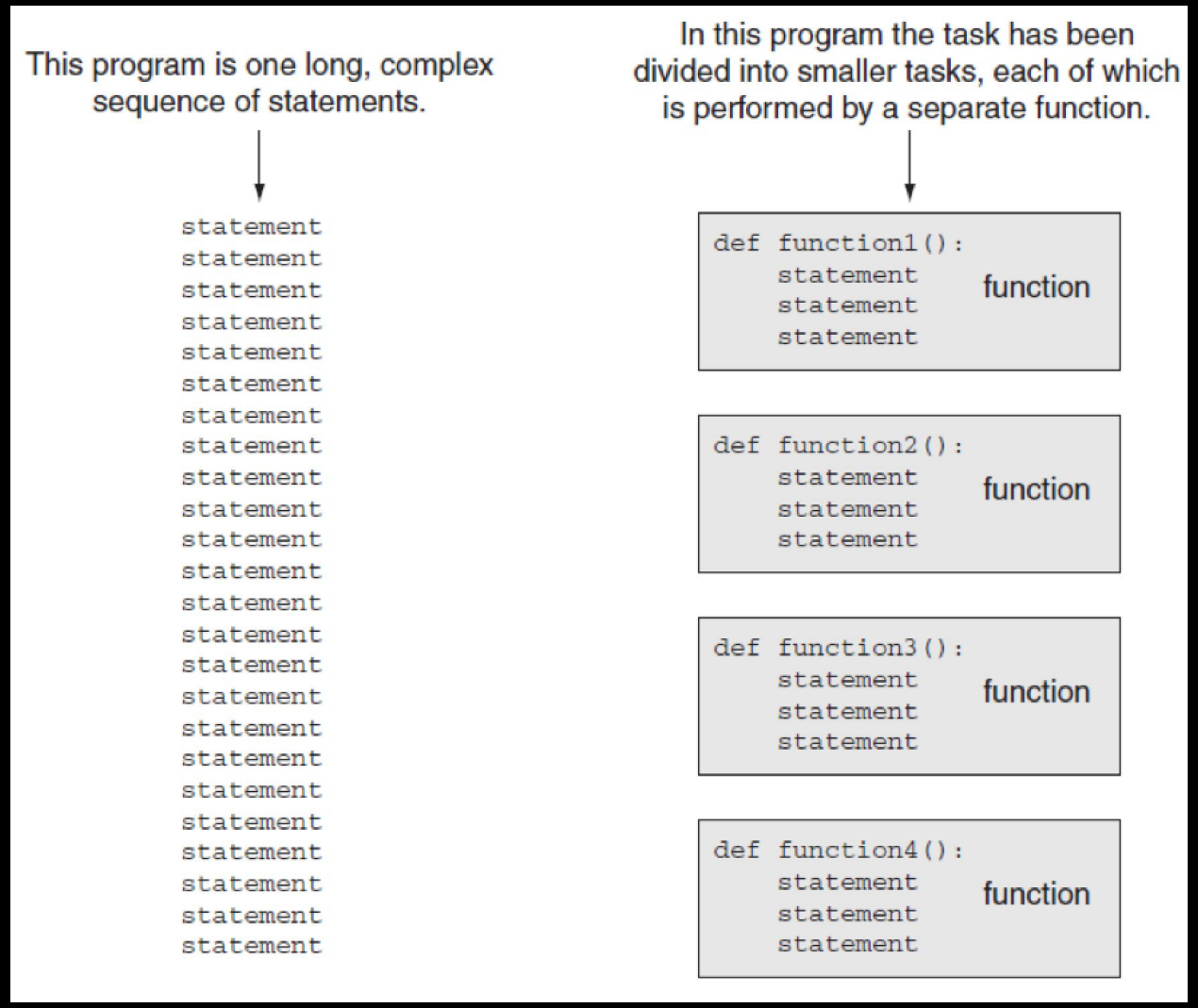
Functions Part I

A function is a group of statements that perform a specific task.

Void functions

Value-returning functions

Functions with arguments (can be void or value-returning)


This program is one long, complex sequence of statements.
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
statement
In this program the task has been divided into smaller tasks, each of which is performed by a separate function.
def function1():
statement
statement
statement
function
def function2():
statement
statement
statement
function
def function3():
statement
statement
statement
function
def function4():
statement
statement
statement
function

Functions: Divide Program into Small Manageable Parts

- → Simpler to understand
- → Eliminates redundancy - Allows code to be written *once* and *reused*, rather than rewritten (copied) in many places - also helpful if fixing errors
- → Faster development for teams - can have common functions used by team members
- → Easier to test: can test each function separately
- → Supports teamwork: team members can work on different parts of program

Procedural Abstraction

Functions, or procedures, also allow for *procedural abstraction.*

Procedural abstraction is when we know what a function or procedure does, or what we want it do, but we do not know how it does it.

This is very useful when working on large, complex, pieces of code with many people often contributing (or even your own code you wrote a few months ago).

Pre-Defined Functions We Have Already Used

print(x, y, z)

int(txt)

float(txt)

input(“Enter a number: ”)

range(start, end, increment)

randint()

Defining Your Own Functions

```
def <function name>(p1, p2, …):
    <statement block>
    return(value)  #only if value returned
```

p1, p2, … are function *parameters*

parameters are not always necessary

Void Functions

A void function does not return any values. (It can still print something though!)

Here is the format of a void function with no arguments:

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

Defining and Calling a Function

```
#defining the function
def example_void_function():
    print(‘Here is an example of a void function’)

#calling the function
example_void_function()
```

Example:
Display Store Menu

```
myBalance = 1000
userChoice = 0
while userChoice !=4:
    print("Welcome to Dons Bank")
    print("What can I do for you?")
    print("1 - Withdraw Funds")
    print("2 - Deposit Funds")
    print("3 - Show Balance")
    print("4 - Quit")
    userChoice = int(input("Please choose one of the above options: "))
    if userChoice == 1:
        amount = int(input("Amount to withdraw: "))
        myBalance = myBalance – amount
        print("New balance: ", myBalance)
    elif userChoice == 2:
    …
```

Example: Function to *Display Store Menu & Get Choice*

```
def showMenuGetChoice():
    print("Welcome to Dons Bank")
    print("What can I do for you?")
    print("1 - Withdraw Funds")
    print("2 - Deposit Funds")
    print("3 - Show Balance")
    print("4 - Quit")
```

Example: Using the Function

```
myBalance = 1000
userChoice = 0
while userChoice !=4:
    showMenuGetChoice()
    userChoice = int(input("Please choose one of the above options: "))
    if userChoice == 1:
        amount = int(input("Amount to withdraw: "))
        myBalance = myBalance – amount
        print("New balance: ", myBalance)
    elif userChoice == 2:
        amount = int(input("Amount to deposit: "))
        myBalance = myBalance + amount
        print("New balance: ", myBalance)
    elif userChoice == 3:
        print("Balance: ", myBalance)
    …
```

main() function

The main() function calls other functions in the program as they are needed.

The main() function can be said to contain the overall logic, or mainline logic, of the program.

We define the main() function with the logic of the overall program, and call it at the end of all function defintions.

```
def main():
    example_void_function()

def example_void_function():
    print(‘Here is an example of a void function called in a main’)

main()
```

An example with main

```
#defining my functions
def message1():
    print('This is message 1')

def message2():
    print('This is message 2')

def message3():
    print('This is message 3')


#calling my functions in the main
def main():
    message1()
    message2()
    message3()


#calling the main()
main()
```

Local and Global Variables

- The ***scope*** of a variable is everywhere it is defined
- Variables created *inside* a function have meaning only inside that function - ***local*** variables
  - The function is their ***scope***
- Variables defined outside any function are called ***global*** variables
  - Their scope includes all functions that declare the global variable

Example of local variable scope

```
def get_name():
    name = input('Enter your name: ') #name is a local variable

def main():
    get_name()
    print('Hello', name)        #Name is out of scope!
                                #Will get error saying name is not defined

main()
```

What would happen if you initialized name outside of the functions? Try running this in Python Tutor Visualize

Example of global variable scope

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

def main():
    global number #using global keyword to declare number variable
                  #because going to assign a value to global variable inside main
    number = int(input('Enter a number: '))
    show_number()

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

main()
```

What would happen if you commented out the line:
global number
?

Functions that take arguments

Some functions take in arguments.

An argument is information that is needed for the function to run. E.g.

```
print(x, y, z)
int(txt)
float(txt)
input(“Enter a number: ”)
range(start, end, increment)
```

Defining and Calling a Function that Takes Arguments

```
def double_number(num):
    result = num*2
    print(result)

double_number(4)
double_number(8)
double_number(1000)
```

```
def main():
    double_number(4)
    double_number(8)

def double_number(num):
    result = num*2
    print(result)

main()
```

Functions with Multiple Arguments

```
def add(num1, num2):
    total = num1 + num2
    print(total)

add(2,3)
```

```
def main():
    add(2,3)
    add(10,20)

def add(num1, num2):
    total = num1 + num2
    print(total)

main()
```