M30299 – Programming
Lecture 08 – Defining functions

Matthew Poole & Nadim Bakhshov
moodle.port.ac.uk

School of Computing
University of Portsmouth

2020/21

Introduction to lecture

- In these two lectures we introduce the idea of **function definitions**.
- This is a first step in allowing us to write larger, more complex, and more useful **programs**.
- In today's lecture we'll cover the the basics of functions and how they work, and will see some simple examples.
- In the next lecture, we'll consider larger case studies.

The idea of functions

- We have already been using function definitions to allow us to write and test many small "programs" within a single file.
- However, most "real-world" programs are longer than those we have written so far.
- Typically, a program is a **collection** of several function definitions.
- The purpose of using functions is:
  - to help **break** a large problem into smaller **parts**;
  - to improve the **readability** of code; and
  - to avoid **repetition**—writing similar code over and over again.
- We will touch on each of these issues here, but will see more realistic examples next lecture and later in the module.

A program as a collection of functions

Commentary

Program

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello()
    sayGoodbye()

main()
```

Screen

A program as a collection of functions

Commentary

Call to function `main`

Program

```
  def sayHello():
      print("Hello")
  def sayGoodbye():
      print("Goodbye")
  def main():
      sayHello()
      sayGoodbye()
> main()
```

Screen

A program as a collection of functions

Commentary

Control transfers to `main`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
      sayHello()
      sayGoodbye()

> main()
```

Screen

A program as a collection of functions

Commentary

Control transfers to `main`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
>     sayHello()
      sayGoodbye()

  main()
```

Screen

A program as a collection of functions

Commentary

Call to function `sayHello`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
>     sayHello()
      sayGoodbye()

  main()
```

Screen

A program as a collection of functions

Commentary

Execution of `main` suspended; control transfers to `sayHello`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
>     sayHello()
      sayGoodbye()

  main()
```

Screen

A program as a collection of functions

Commentary

Execution of `main` suspended; control transfers to `sayHello`

Program

```
  def sayHello():
>     print("Hello")
  def sayGoodbye():
      print("Goodbye")
  def main():
      sayHello()
      sayGoodbye()
  main()
```

Screen

A program as a collection of functions

Commentary

Displays “Hello”

Program

```
  def sayHello():
>     print("Hello")
  def sayGoodbye():
      print("Goodbye")
  def main():
      sayHello()
      sayGoodbye()
  main()
```

Screen

A program as a collection of functions

Commentary

Execution of `sayHello` completed; control returns to `main`

Program

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello()
    sayGoodbye()

main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Execution of `sayHello` completed; control returns to `main`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
      sayHello()
>     sayGoodbye()

  main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Call to function `sayGoodbye`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
      print("Goodbye")

  def main():
      sayHello()
>     sayGoodbye()

  main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Execution of `main` suspended; control transfers to `sayGoodbye`

Program

```
  def sayHello():
      print("Hello")
  def sayGoodbye():
      print("Goodbye")
  def main():
      sayHello()
>     sayGoodbye()
  main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Execution of `main` suspended; control transfers to `sayGoodbye`

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
>     print("Goodbye")

  def main():
      sayHello()
      sayGoodbye()

  main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Displays "Goodbye"

Program

```
  def sayHello():
      print("Hello")

  def sayGoodbye():
>     print("Goodbye")

  def main():
      sayHello()
      sayGoodbye()

  main()
```

Screen

```
Hello
```

A program as a collection of functions

Commentary

Execution of `sayGoodbye` completed; control returns to `main`

Program

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello()
    sayGoodbye()

main()
```

Screen

```
Hello
Goodbye
```

A program as a collection of functions

Commentary

Execution of `sayGoodbye` completed; control returns to `main`

Program

```
def sayHello():
    print("Hello")
def sayGoodbye():
    print("Goodbye")
def main():
    sayHello()
    sayGoodbye()
main()
```

Screen

```
Hello
Goodbye
```

A program as a collection of functions

Commentary

Execution of `main` completed.

Program

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello()
    sayGoodbye()

main()
```

Screen

```
Hello
Goodbye
```

A program as a collection of functions

Commentary

Execution of program completed

Program

```
def sayHello():
    print("Hello")

def sayGoodbye():
    print("Goodbye")

def main():
    sayHello()
    sayGoodbye()

main()
```

Screen

```
Hello
Goodbye
```

Using functions to break down large problems

- Clearly, to write a program to carry out such a simple task we don't really need to use several functions.
- However, let's briefly consider a more complicated problem:

  Write a program that reads data about employees (salary, overtime hours, etc.) from a file, and then displays how much each should be paid this month.
- A first step in **designing** a solution to this problem might be to break the problem down into three simpler **sub-problems**:
  - read employees' data;
  - calculate wages; and
  - display wages.

Using functions to break down large problems

- Each could be solved using a function, and `main` would be:

```
def main():
    ... readEmployees(...)
    ... calculateWages(...)
    ... displayWages(...)
```

- It is possible that the sub-problems could be broken down further. E.g., to calculate the wages of an employee involves:
  - calculating basic pay;
  - calculating overtime pay; and
  - deducting tax.
- The solution to these sub-problems might be functions themselves.

Functions with parameters

- Imagine that we want to display greetings to several different people. We might write the following:

```
def main():
    print("Hello, Vicky. How are you today?")
    print("Hello, Tom. How are you today?")
    print("Hello, Fred. How are you today?")
    print("Hello, Sam. How are you today?")
    print("Hello, Gemma. How are you today?")
```

- This would work, but it contains a lot of repeated code: the only difference between the greetings is the person's name.
- We define a function that has the name as a **parameter**...

Functions with parameters

```
def greet(name):
    print("Hello " + name + ".", end=" ")
    print("How are you today?")
```

- The parameter `name` is a special variable whose value is initialised when the function is **invoked** or **called**.
- When we call this function we supply an **argument** (a value for the parameter):

```
>>> greet("Sam")
Hello Sam. How are you today?
>>> greet("Fred")
Hello Fred. How are you today?
```

Functions with parameters

- Our `main` function can now be replaced:

```
def main():
    greet("Vicky")
    greet("Tom")
    greet("Fred")
    greet("Sam")
    greet("Gemma")
```

- Apart from the overall reduction in text, what other advantage(s) might this new code – `greet` & `main` – have?

Functions with parameters - operation

Commentary

Assume function `main` has been called

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Call to function `greet` with argument `"Vicky"`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
>   greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Parameter `name` set as if using an assignment `name = "Vicky"`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
>   greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Parameter `name` set as if using an assignment `name = "Vicky"`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name $\boxed{\text{"Vicky"}}$

Caller

```
def main():
>   greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Execution of `main` suspended; control passed to `greet`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name $\boxed{\text{"Vicky"}}$

Caller

```
def main():
>   greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Execution of `main` suspended; control passed to `greet`

Callee

```
def greet(name):
>   print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name	"Vicky"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Displays `Hello, Vicky.`

Callee

```
def greet(name):
>    print("Hello" + name + ".", end=" ")
     print("How are you today?")
```

Variables

name	"Vicky"

Caller

```
def main():
     greet("Vicky")
     me = "Sam"
     greet(me)
```

Variables

Functions with parameters - operation

Commentary

Displays `How are you today?`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
>   print("How are you today?")
```

Variables

name "Vicky"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Execution of `greet` completed; control returned to `main`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name | "Vicky"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Execution of `greet` completed; control returned to `main`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
>   me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Assigns `me` the value `"Sam"`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
>   me = "Sam"
    greet(me)
```

Variables

Functions with parameters - operation

Commentary

Assigns `me` the value `"Sam"`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
>   me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Call to function `greet` with argument `me`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
>   greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Parameter `name` set as if using an assignment `name = me`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
>   greet(me)
```

Variables

me $\boxed{\text{"Sam"}}$

Functions with parameters - operation

Commentary

Parameter `name` set as if using an assignment `name = me`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name "Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
>   greet(me)
```

Variables

me "Sam"

Functions with parameters - operation

Commentary

Execution of `main` suspended; control passed to `greet`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name	"Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
>   greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Execution of `main` suspended; control passed to `greet`

Callee

```
def greet(name):
>   print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name	"Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Displays `Hello, Sam.`

Callee

```
def greet(name):
>   print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name	"Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Displays `How are you today?`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
>   print("How are you today?")
```

Variables

name	"Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Execution of `greet` completed; control returned to `main`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

name "Sam"

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me "Sam"

Functions with parameters - operation

Commentary

Execution of `greet` completed; control returned to `main`

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Execution of `main` completed

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

me	"Sam"

Functions with parameters - operation

Commentary

Execution of `main` completed

Callee

```
def greet(name):
    print("Hello" + name + ".", end=" ")
    print("How are you today?")
```

Variables

Caller

```
def main():
    greet("Vicky")
    me = "Sam"
    greet(me)
```

Variables

Functions that return values

- We have used built-in functions that **return** values to the caller:

```
>>> euros = float(input("Enter amount in euros: "))
>>> print(math.sqrt(2))
1.4142135623730951
```

- Notice that these function calls are **expressions** (they have **values** and often appear on the right-hand-side of assignments).
- Let's write our own function that returns values:

```
>>> def square(x):
        return x * x
```

Functions that return values

- A **return statement** like this causes:
    - the function to exit (control is passed back to the caller), and:
    - the returned value is given as the function call's value.
- Let's use our user-defined `square` function:

```
>>> print(square(2))
4
>>> z = 3
>>> y = square(z)
>>> print(y)
9
```

Functions that return multiple values

- We note that functions can return more than one value; e.g:

```
>>> def sumDiff(n1, n2):
        return n1 + n2, n1 - n2
```

returns both the sum and the difference of two numbers.

- Such functions are often used together with a **simultaneous assignment**:

```
>>> x, y = 2, 10
>>> x
2
>>> y
10
```

```
>>> x, y = y + 5, x
>>> x
15
>>> y
2
```

Functions that return multiple values

- We can thus take the two values returned by

```
>>> def sumDiff(n1, n2):
        return n1 + n2, n1 - n2
```

  and assign them to two separate variables:

```
>>> s, d = sumDiff(10, 3)
>>> s
13
>>> d
7
```

Functions cannot change argument values

Commentary

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

Functions cannot change argument values

Commentary

Assign `temperature` the value 15

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
  def main():
>     temperature = 15
      turnUpHeat(temperature)
      print(temperature)
```

Variables

Functions cannot change argument values

Commentary

Assign `temperature` the value 15

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
  def main():
>     temperature = 15
      turnUpHeat(temperature)
      print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Call to function `turnUpHeat` with argument `temperature`

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
  def main():
      temperature = 15
>     turnUpHeat(temperature)
      print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Parameter `temp` set (`temp = temperature`)

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
 def main():
     temperature = 15
>    turnUpHeat(temperature)
     print(temperature)
```

Variables

temperature	15

Functions cannot change argument values

Commentary

Parameter `temp` set (`temp = temperature`)

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

temp $\boxed{15}$

Caller

```
 def main():
     temperature = 15
>    turnUpHeat(temperature)
     print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Execution of `main` suspended; control passed to `turnUpHeat`

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

temp $\boxed{15}$

Caller

```
 def main():
     temperature = 15
>    turnUpHeat(temperature)
     print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Execution of `main` suspended; control passed to `turnUpHeat`

Callee

```
  def turnUpHeat(temp):
>     temp = temp + 10
```

Variables

temp $\boxed{15}$

Caller

```
  def main():
      temperature = 15
      turnUpHeat(temperature)
      print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Increase value of `temp` by 10

Callee

```
 def turnUpHeat(temp):
>     temp = temp + 10
```

Variables

temp $\boxed{15}$

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Increase value of `temp` by 10

Callee

```
 def turnUpHeat(temp):
>     temp = temp + 10
```

Variables

temp $\boxed{25}$

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Execution of `turnUpHeat` completed; control returned to `main`

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

temp $\boxed{25}$

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

temperature $\boxed{15}$

Functions cannot change argument values

Commentary

Execution of `turnUpHeat` completed; control returned to `main`

Callee

```
 def turnUpHeat(temp):
     temp = temp + 10
```

Variables

Caller

```
 def main():
     temperature = 15
     turnUpHeat(temperature)
>    print(temperature)
```

Variables

temperature	15

Functions cannot change argument values

Commentary

Display values of `temperature` (15)

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
  def main():
      temperature = 15
      turnUpHeat(temperature)
>     print(temperature)
```

Variables

temperature	15

Functions cannot change argument values

Commentary

Execution of `main` completed

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

temperature	15

Functions cannot change argument values

Commentary

Execution of `main` completed

Callee

```
def turnUpHeat(temp):
    temp = temp + 10
```

Variables

Caller

```
def main():
    temperature = 15
    turnUpHeat(temperature)
    print(temperature)
```

Variables

Functions cannot change argument values

- To cause a change of temperature we should rewrite our code using a function that **returns** a new temperature:

```
def hotterTemp(temp):
    return temp + 10
def main():
    temperature = 15
    temperature = hotterTemp(temperature)
    print(temperature)
```

- We see also that:
  - the call `turnUpHeat(temperature)` is a **statement**; and
  - the call `hotterTemp(temperature)` is an **expression**.