MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT9133 Semester 1 2019
Programming Foundations in Python

Week 5:
Decompositions, Functions, and Modules

Chunyang Chen
Gavin Kroeger

GROUP OF EIGHT AUSTRALIA

ALERT!

WARNING

If you are reading these slides rather than watching/attending the lecture (shame on you) then you will be underwhelmed by the content here!

Most of this week's lesson is done as demonstration, so watch the lecture recording.

If for some reason the recording is corrupted or fails, let me know ASAP and I will fix it.

Midsemester Test

- Takes place in Week 8, 16th of September
- No lecture that week, the lecture session will be broken up into two hour slots (10am-11am and 11am-12pm).
- Students will be assigned to a particular slot
- I am working on getting a sample (its harder than you think to get)

Midsemester Test

- If you cannot attend, you must make arrangements with us.
- You must give a reason (wanting to stay at home is not a reason)
- Email the role account: fit9133.allcampuses-x@monash.edu
- Please give it an appropriate subject line and include your reason with your student ID in the main text

Module 3 Synopsis

- Module 3 is aimed to introduce you with:
  - Concepts of decomposition
    - Functions and methods
    - Modules in Python
  - Concepts of classes and methods
    - Implementation
    - Object instantiation
    - Overloading methods

Module 3 Learning Objectives

- Upon completing this module, you should be able to:
  - Identify how to decompose a computational program into manageable units of functions and/or classes.
  - Understand the basic concepts of object oriented programming.

MONASH
University

Concepts of Decomposition: Functions and Python Modules

Concepts of Decomposition

- Decomposition:
  - One of problem solving strategies
  - A process of breaking a complex problem into simpler, independent, manageable units
  - Easier to conceive, understand, solve, and program

Concepts of Decomposition

- Decomposition in programming:
  - *Understand the input and the output of each component*

- Components in programming
  - Functions
  - Classes (and methods)
  - Modules (packages) in Python

Functions

- Definition of functions:
  - An independent named set of statements
  - Defined to achieve a specific task for solving a problem
  - Take any number of *arguments* (parameters) of any data type
  - Return any number of results of any data type (*optional*)
  - Create a function and call or invoke a function

- Functions as a form of *abstraction*:
  - Procedural or functional abstraction
  - Users only need to know how to put together all the available functions to solve a problem (without worrying about the underlying implementation details)

(More on) Functions

- Implementation of functions:
  - Defined with the keyword **`def`**
  - Function header: function name and parameters (arguments)
  - Return type(s) are not required

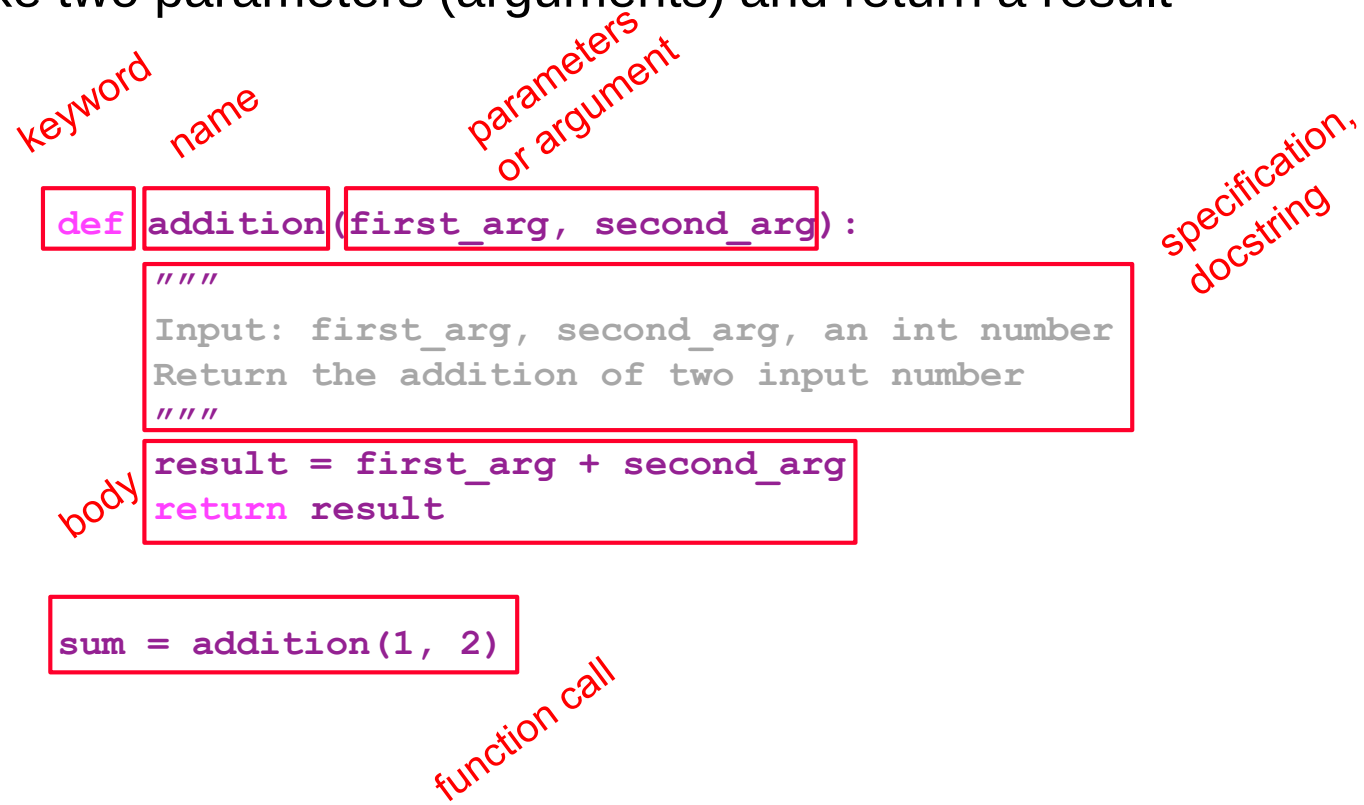
- Function characteristics:
  - has a name
  - has parameter (0 or more)
  - has a docstring (optional but recommended)
  - has a body
  - returns something

(More on) Functions

- Naming convention for functions:
  - Similar to variables names
  - camelCase style or using a single underscore as the delimiter between words
  - Start with a letter or an underscore; must contain only letters, numbers, and the underscore

Example: A Simple Function

- Defining a function (without default values):
  - Take two parameters (arguments) and return a result



Example: A Simple Function

- Defining a function (without default values):
  - Take two parameters (arguments) and return a result

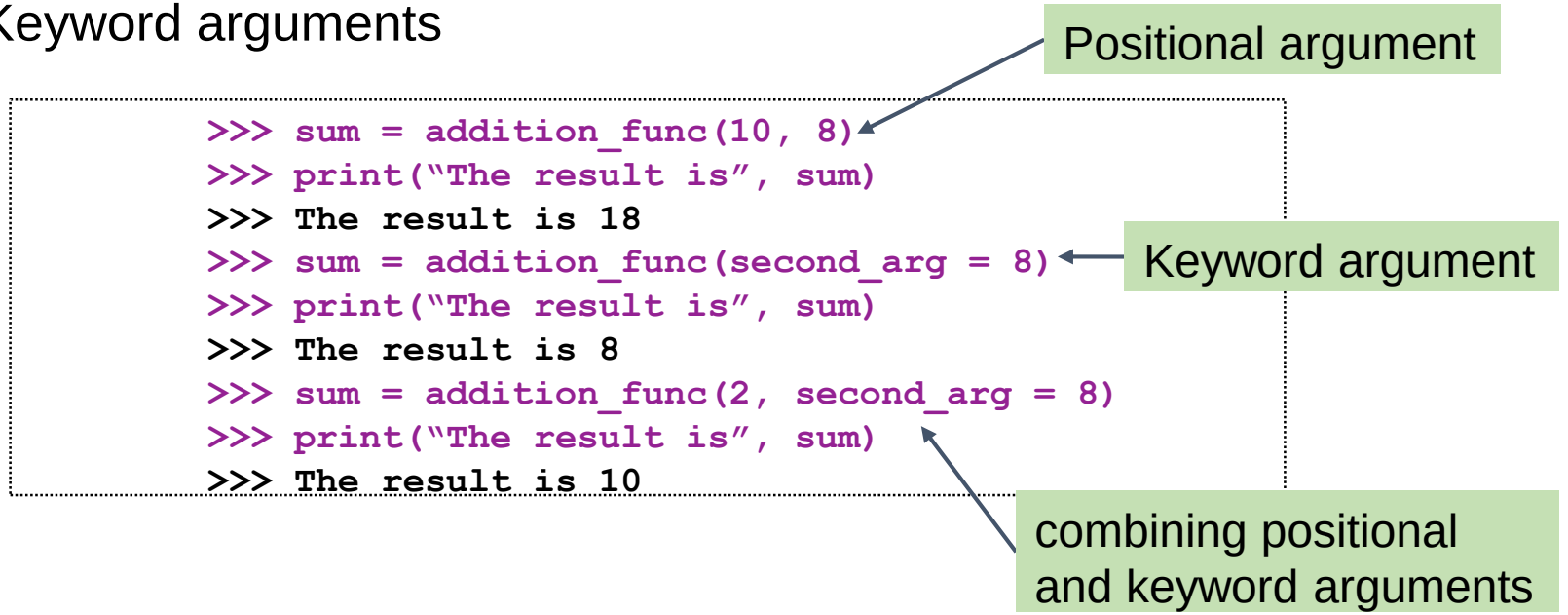
```
def addition_func(first_arg, second_arg):
    result = first_arg + second_arg
    return result
```

- Defining a function (with default values):
  - Parameters (arguments) are assigned with default values

```
def addition_func(first_arg = 0, second_arg = 0):
    result = first_arg + second_arg
    return result
```

Example: A Simple Function

- Invoking or calling a function:
  - Positional arguments
  - Keyword arguments



Review Question 1

Which of the following is a *valid* function call (invocation)?

```
def product_func(first_arg, second_arg):
    return first_arg * second_arg
```

A. product = product_func()
B. product = product_func(3)
C. product = product_func(3,6)
D. product = product_func(,6)
E. Not sure

Difference between Return and Print

return	vs.	print
• Return only has meaning inside a function • Only one return executed inside a function • Code inside function but after return statement not executed • Has a value associated with it, given to function caller		• print can be used outside functions • Can execute many print statement inside a function • Code inside function can be executed after a print statement • Has a value associated with it, outputted to the console

The Main Function

- **`main()`**:
  - Specify the flow of execution of a program
  - Define how functions are invoked in a specific order within a program

```
def function1():
    ...

def function2():
    ...

def function3():
    ...

def main():
    function2()
    function3()
    function1()

if __name__ == "__main__":
    main()
```

Example: The Main Function

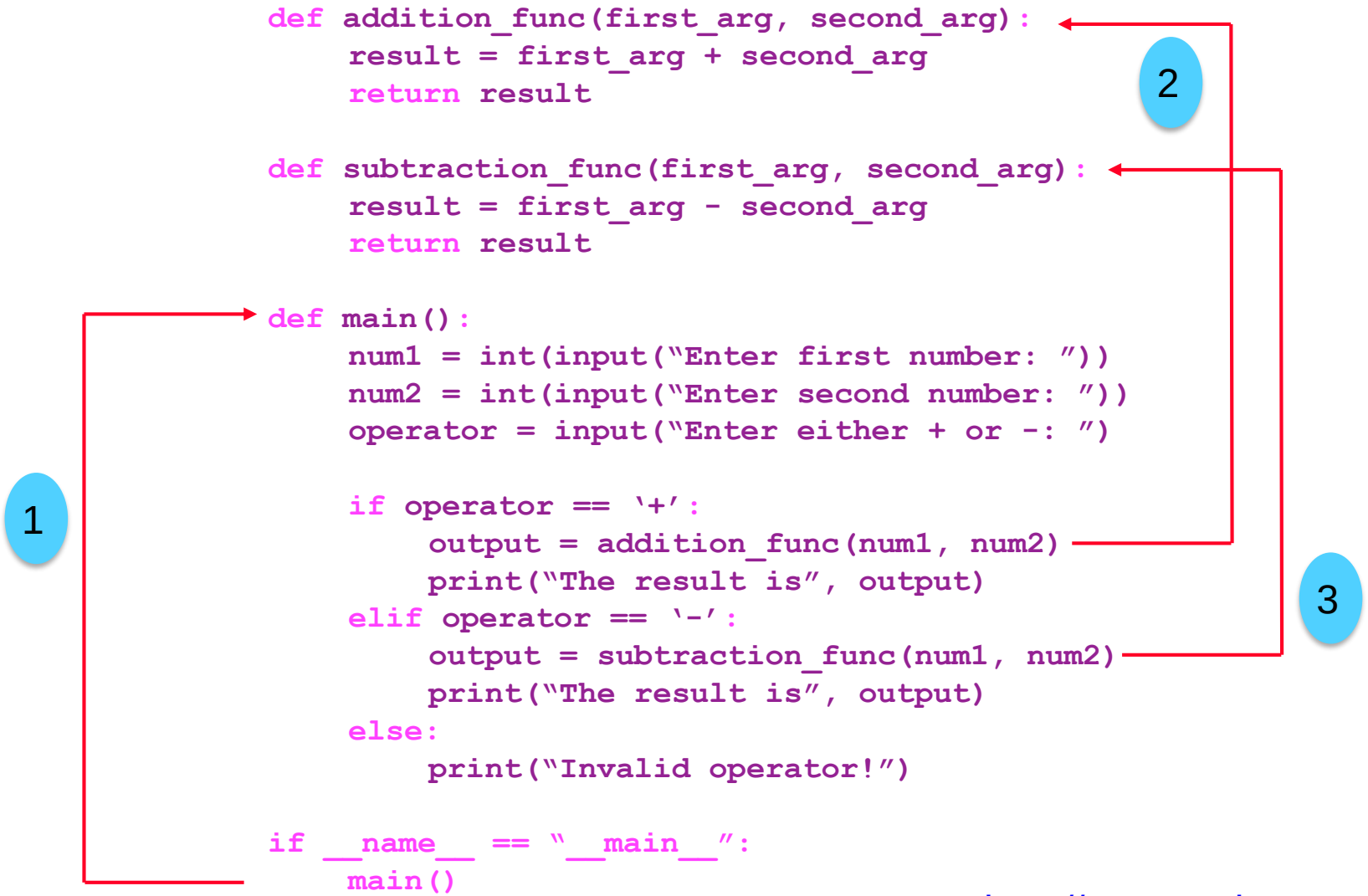
```
def addition_func(first_arg, second_arg):
    result = first_arg + second_arg
    return result

def subtraction_func(first_arg, second_arg):
    result = first_arg - second_arg
    return result

def main():
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    operator = input("Enter either + or -: ")

    if operator == '+':
        output = addition_func(num1, num2)
        print("The result is", output)
    elif operator == '-':
        output = subtraction_func(num1, num2)
        print("The result is", output)
    else:
        print("Invalid operator!")

if __name__ == "__main__":
    main()
```



http://www.pythontutor.com/

Modules in Python

- Modules in Python:
  - Source files containing Python definitions and statements
  - Intended to be used by other Python programs
  - Each module can contain: function definitions and/or class definitions (as well as method definitions)

- To use a module (with `.py` extension) in another Python program:
  - Use the keyword `import`
  - Syntax: `import <module_name>` or
    `import <module_name> as <alias>`

- To use a function from a given module:
  - Syntax: `<module_name>.<function_name>` or
    `<alias>.<function_name>`

Example: Modules in Python

```
# Module name: arithmetic.py
# Description: Defined four arithmetic functions

def addition_func(first_arg, second_arg):
    result = first_arg + second_arg
    return result

def subtraction_func(first_arg, second_arg):
    result = first_arg - second_arg
    return result

def multiplication_func(first_arg, second_arg):
    result = first_arg * second_arg
    return result

def division_func(first_arg, second_arg):
    result = first_arg // second_arg
    return result
```

Example: Modules in Python

```
import arithmetic as arm

def main():
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    operator = input("Enter +, -, *, or /: ")

    if operator == '+':
        output = arm.addition_func(num1, num2)
        print("The result is", output)
    elif operator == '-':
        output = arm.subtraction_func(num1, num2)
        print("The result is", output)
    elif operator == '*':
        output = arm.multiplication_func(num1, num2)
        print("The result is", output)
    elif operator == '/':
        output = arm.division_func(num1, num2)
        print("The result is", output)
    else:
        print("Invalid operator!")

if __name__ == "__main__":
    main()
```

Review Question 3

Given the following Python module named **`arithmetic.py`**, which of the following is a valid function call (invocation)?

```
def addition_func(first_arg=1, second_arg=2):
    return first_arg + second_arg

def subtraction_func(first_arg=1, second_arg=2):
    return first_arg - second_arg
```

A. import arithmetic.py
   result = arithmetic.addition_func()
B. import arithmetic
   result = addition_func()
C. import arithmetic
   result = arithmetic.addition_func()
D. Not sure

Why function?

- DRY principle
  - Don’t Repeat Yourself
  - Reducing repetition of software patterns
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code

Week 5 Summary

- So far, we have discussed:
  - Concepts of decomposition
  - Functions

- Next week:
  - Classes (and methods)
  - Variable scoping and lifetime

Reminder: Assignment 1 due on Sunday (8 Sep) 11:55pm.