Introduction to programming using Python

Session 4

Matthieu Choplin

http://pythonguy.com/

Objectives of Session 4

- Remainder of Session 3: Quiz
- Getting to know Pycharm
- Functions
- Modularity
- Debugging using the pdb library

Remainder of Session 3: Quiz (1)

- What is the difference between the keywords *break* and *continue*?

Remainder of Session 3: Quiz (2)

- What is the difference between range(10), range(0, 10), and range(0, 10, 1) in a for loop?

Pycharm

- To execute python scripts from Pycharm, you need to define an interpreter: File/Settings/Project/Project Interpreter

Built in functions seen so far

Input/Ouput	**Conversion type:**	**Introspection:**
input()	int()	type()
print()	float()	

All the built in functions:
https://docs.python.org/3.6/library/functions.html

Defining our own function

To define a function, we use the keyword **def**, the name of the function, the brackets, and the colon

Then the body of the function needs to be indented

```
def name_of_the_function():
    # body of the function
```

When we define a function, we just make python see that the function exist but it is not executed

```
def my_function():
  print("THIS IS MY FUNCTION")
```

Calling our own function

To call or execute or run a function, we use the name of the function AND the brackets, without the brackets, the function is not called.

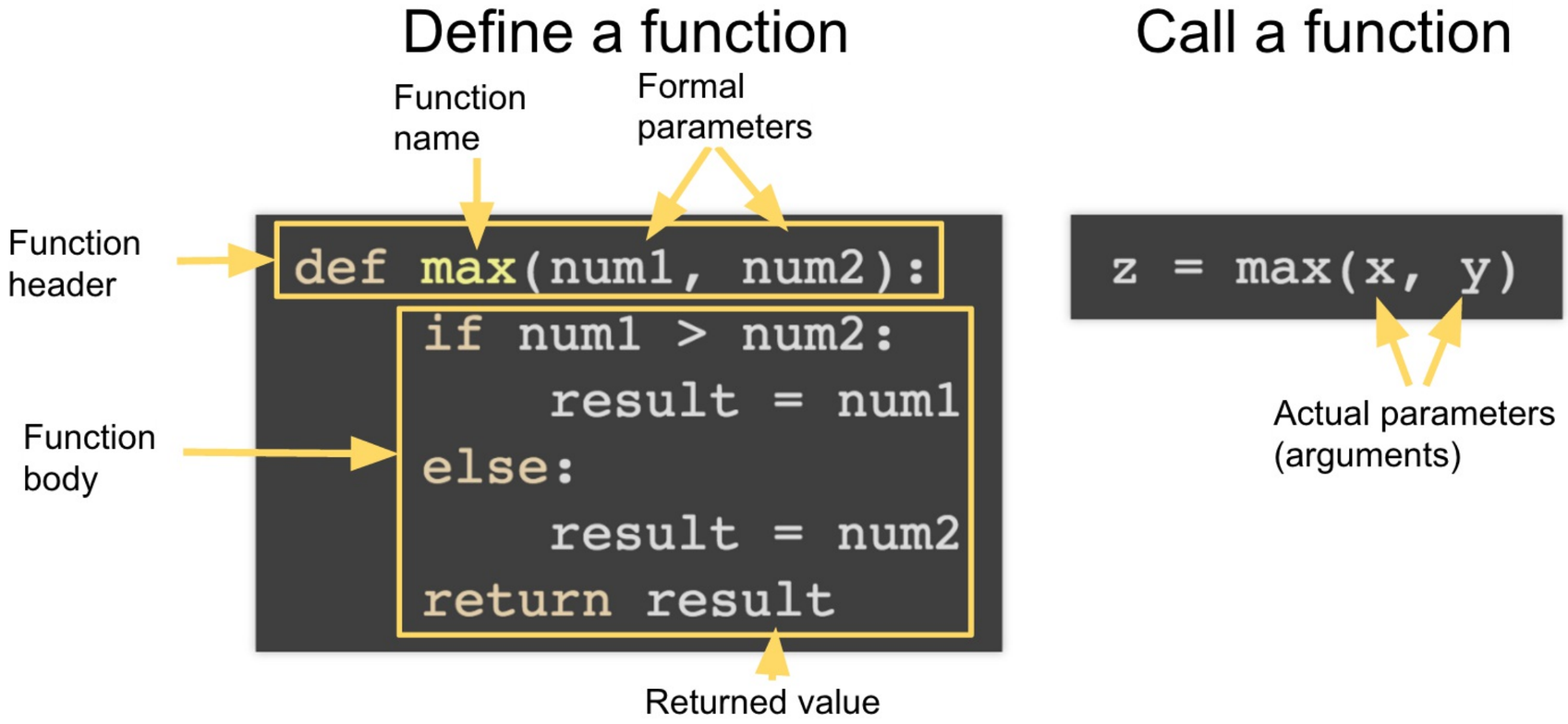
```
name_of_the_function()
```

Notice the difference between defining and calling a function

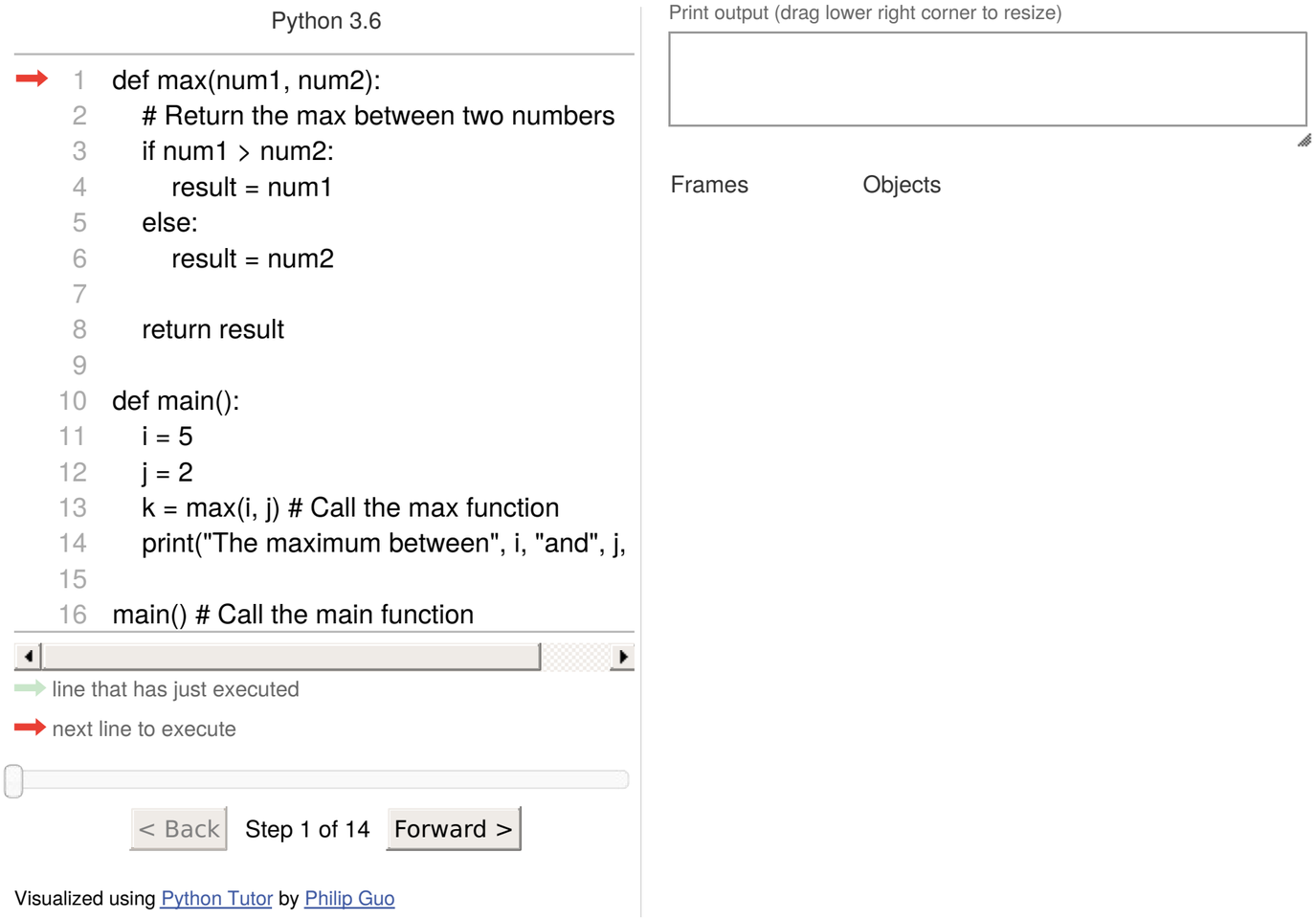
```
def my_function():
  print("THIS IS MY FUNCTION")

my_function()
```

Defining and Calling Functions

A function is a collection of statements that are grouped together to perform an operation.



How a function gets called



Functions With/Without Return Values

- A function with the **return** keyword explicitly return a **value**. For example the function max() in the previous program.
- A function **does something** but does not return a value. For example the function main() in the previous program.

Example of a function that does something without returning a value

```
def printGrade(score):
    # Print grade for the score
    if score >= 90.0:
        print('A')
    elif score >= 80.0:
        print('B')
    elif score >= 70.0:
        print('C')
    elif score >= 60.0:
        print('D')
    else:
        print('F')

def main():
    score = eval(input("Enter a score: "))
    print("The grade is ", end = "")
    printGrade(score)

main() # Call the main function
```

Example of a function that returns a value

```
def getGrade(score):
    # Return the grade for the score
    if score >= 90.0:
        return 'A'
    elif score >= 80.0:
        return 'B'
    elif score >= 70.0:
        return 'C'
    elif score >= 60.0:
        return 'D'
    else:
        return 'F'

def main():
    score = eval(input("Enter a score: "))
    print("The grade is", getGrade(score))

main() # Call the main function
```

The None Value

A function that does not return a value is known as a void function. In Python, such function returns a special None.

```
def sum(number1, number2):
    total = number1 + number2

print(sum(1, 3))
```

Passing Arguments by Positions

Suppose you have the following function:

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

What is the output of **nPrintln("Welcome to Python", 5)**?

What is the output of **nPrintln(15, "Computer Science")**?

What is wrong? How to fix?

Keyword Arguments

With the same function:

```
def nPrintln(message, n):
    for i in range(0, n):
        print(message)
```

What is the output of **nPrintln(message="Welcome to Python", n=5)**

What is the output of **nPrintln(n = 4, message = "Computer Science")**

What is wrong? How to fix?

Default Arguments

Python allows you to define functions with default argument values. The default values are passed to the parameters when a function is invoked without the arguments.

```
def printArea(width = 1, height = 2):
    area = width * height
    print("width:", width, "\theight:", height, "\tarea:", area)

printArea() # Default arguments width = 1 and height = 2
printArea(4, 2.5) # Positional arguments width = 4 and height = 2.5
printArea(height = 5, width = 3) # Keyword arguments width
printArea(width = 1.2) # Default height = 2
printArea(height = 6.2) # Default widht = 1
```

Modularizing Code (1)

We can make use of existing builtin modules/libraries

For example, you can use the *random* library

To get the list of all available modules, checkout the *help()* interactive function

Exercise: Guess Number

Make a program to ask the user to guess the number that has been randomly generated.

Start from this file: GuessNumber.py (right click and save as)

- The user will be able to try continuously until he finds the correct number.
- The program will stop as soon as the number is found, i.e. as soon as the random number matches the entered number
- At each iteration, i.e. each time the user tries a number and presses enter, the program will say if the number is too high, too low or correct

Solution: Guess Number

- Show solution

Modularizing Code (2)

Functions can be used to reduce redundant coding and enable code reuse. Functions can also be used to modularize code and improve the quality of the program.

A python file is called a module, you can import a module

Example, download the following files and put them in the same directory (right click and save as):

- GCDFunction.py
- TestGCDFunction.py

Exercise: Use the isPrime Function

The program PrimeNumberFunction.py (right click and save as) provides the isPrime(number) function for testing whether a number is prime.

Use this function to find the number of prime numbers less than 10,000.

- Reuse the function in the same file
- Import the function in an other file

Using the function in the same file

- Solution

Import the function from an other file

- Solution

Using a check to avoid executing the script when we import it

```
def bla():
    print('bla')

if __name__ == '__main__':
    bla()
```

This is checking if the file is executed directly or if it imported

Download these 2 files: ex.py and some_lib.py(right click and save as) and place them in the same directory

Debugging

- What is the program supposed to do?
- Is it doing what it is expected to do?
- Why not? Investigate...

2 ways of debugging

- Naive debugging
  - Use the **print()** function, sometimes it is enough
- Smarter debugging
  - Use a debugger, i.e. **pdb** and insert a **breakpoint**
  - A breakpoint is an intentional stopping or pausing place in a program. It is also sometimes simply referred to as a pause.
  - You set it by writing the following within your program

```
import pdb; pdb.set_trace()
```

Commands for using pdb

- list (**l**) List 11 lines around the current line (five before and five after). Using list with a single numerical argument lists 11 lines around that line instead of the current line.
- next (**n**) Execute the next line in the file. This allows you to go line by line and inspect the state of the code at that point.
- continue (**c**) Exit out of the debugger but still execute the code.
- step into (**s**) to go into the execution call of an other function

To go further: https://pymotw.com/3/pdb/

Documentation Strings (Docstring)

```
def my_function():
    """Do nothing, but document it.
    No, really, it doesn't do anything.
    """
    pass

help(my_function)
```