Programming with Python: Functions

- Programming -
Mechanical and Electrical Engineering

Carlos III University of Madrid

uc3m | Universidad Carlos III de Madrid
Departamento de Informática

Introduction (I)

- So far, we've been dealing with the implementation of programs in an isolated way
  - Program to determine whether a given a number is prime, program to determine a position of the Fibonacci sequence, etc.
- What could we do if we wanted to reuse some code that we've implemented, among different programs or in different parts of a same program, without having to copy & paste the code?
  - E.g. we might want to reuse code that generates a random integer number in a given range
- We use **functions** to achieve that.

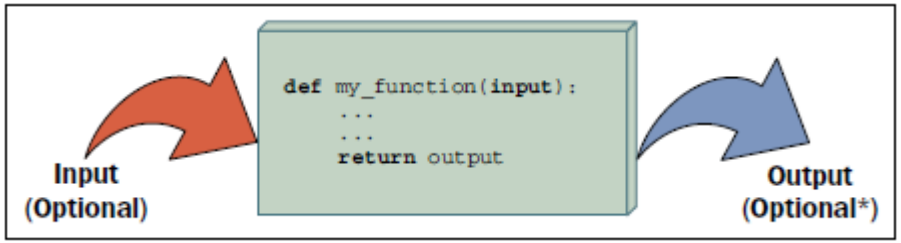
Introduction (II)

- The general purpose of a function is to provide an output based on some input
- Functions...
  - provide a general mechanism that enables us to reuse code
  - allow us to clearly separate tasks within a program
  - improves readability
  - split a complex task in small activities
  - allow us to transfer control back and forth between different pieces of code
    - Functions are called from other pieces of code; it's as if the function code was added to the "caller" code
- We've already been using functions during the course
  - E.g. len, zip, print…

Function Declaration

- A function declaration contains the code that defines the behaviour enacted when the function is called
  - A method encapsulates recurring behaviour
- A method declaration consists of **signature and body**
- Example:

```
def function(input_variables) :
    # code here
    return output_variable
```



Functions

Some examples of functions are:

- Calculates the perimeter of a circle
- Changes the vowels of a String for ‘x’
- Indicates whether a String is a palindrome
- Prints to screen a String in capital letters
- Indicates the highest double number from a set of 5
- Indicates the highest double number in an array
- Returns the nth value of the Fibonacci sequence
- Indicates whether a matrix is diagonal
- Returns the transpose of a matrix

Variable Scope (I)

- The variables in a Python program have a specific scope in which they can be accessed
- As a rule of thumb, and for this course, variables should be accessible in the code block corresponding to the most immediate open execution block
- This means that e.g.:
  - Variables are not accessible between functions
  - When you declare a variable within the block of instructions of a loop (including for initialization), the variable isn't (*shouldn't be*) accessible outside the loop

Variable Scope (II)

- The variables declared in a method are referred to as local variables
- Method arguments can be used as local variables
- Trying to access a variable outside its scope will result in a compilation error
- The variables within a same scope must have different names
  - E.g. an argument of a method cannot have the same name as a local variable of the method

Example

- Lets study the scope and name resolution using an example:

```
def my_function():
    test = 1 # this is defined in the local scope of the function
    print("my_function:", test)

test = 0 # this is defined in the global scope
my_function()
print("global:", test)
```

The name test is defined in two different places. Actually, two scopes.

- If you execute the code, you'll see this:

  my_function: 1
  global: 0

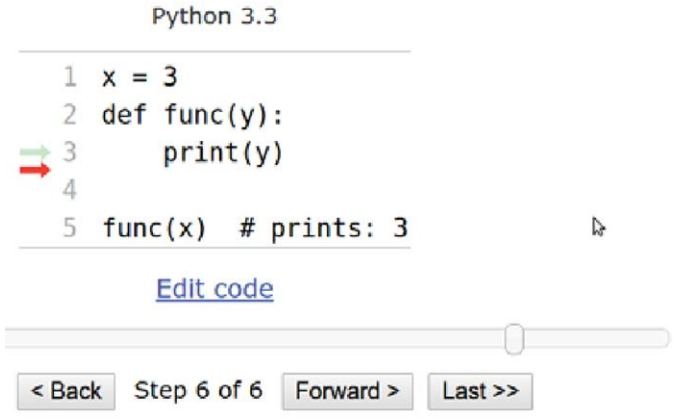
Input: Argument passing

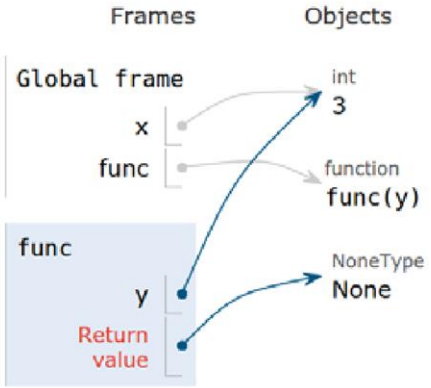
- There are three key points to keep in mind:
  - Argument passing is nothing more than assigning an object to a local variable name
  - Assigning an object to an argument name inside a function doesn't affect the caller
  - Changing a mutable object argument in a function affects the caller

Input: Argument passing (Example)

key.points.argument.passing.py

```
x = 3
def func(y):
    print(y)
func(x)                 # prints: 3
```





Return values

- Return values of functions are one of those things where Python is light years ahead of most other languages.
- Functions are usually allowed to return one object (one value) but, in Python, you can return a tuple, and this implies that you can return whatever you want.

Return values

- Example:

```
def toHMS(s):
    hours = s // 3600
    minutes = (s % 3600) // 60
    seconds = (s % 3600) % 60
    return (hours, minutes, seconds)

x = int(input("Enter a number of seconds:"))

(h, m, s) = toHMS(x)
print(x,"seconds are:", h, "hours, ", m, "minutes, ", s, "seconds")
```

None value

- A function always returns something in Python, even if you don't explicitly use the *return* clause.
- If the function has no return statement in the body, its return value is *None*.
- *None* defines a *null* value - or the absence of value. *None* is **not** the same as *0*, *0.0*, *False* or an empty String (*""*)
- *None* is a data type of its own.

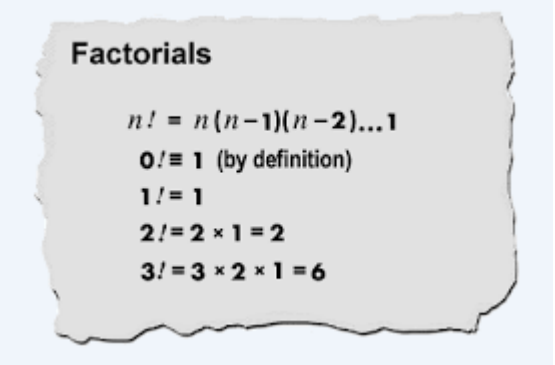
```
def printHello():
    print("Hello world")

x = printHello()
print(x)
```

```
Hello world
None
```

Recursive functions

- When a function calls itself to produce a result, it is said to be **recursive**. Sometimes recursive functions are very useful in that they make it easier to write code.
- Example:

```
def factorial(n):
    if n in (0, 1): # base case
        return 1
    return factorial(n - 1) * n # recursive case
```



Documenting the code

- Using triple double-quoted strings allows you to expand easily later on.
- Use sentences that end in a period, and don't leave blank lines before or after.
- Multi-line comments are structured in a similar way.
- Example:

```
def square(n):
    """Return the square of a number n. """
    return n ** 2
def get_username(userid):
    """Return the username of a user given their id. """
    return db.get(user_id=userid).username
```

Conclusion

- A function is a piece of code that generates an output from some input
  - It is a reusable task within a program
- A method has a signature and body
- The scope of a variable corresponds to a code block
  - Generally, it starts from the place where it is declared to the closing bracket of its code block
- In general, whenever you can clearly separate tasks within programs, you should do so
- Document your code!