COMP 10280
Programming I (Conversion)

John Dunnion

School of Computer Science
University College Dublin

COMP 10280 Programming I (Conversion)/Lecture 14

Outline

Functions (1)

- We have developed a number of programs so far
- While the code is useful, it is limited in its general use
- If we want to reuse the code, we have to copy the code, possibly edit it to change variable names, constants, etc
- We would like to be able to use our code again and in more complex programs

Functions (2)

- For example, consider our program to calculate the cube root of a number
- If we wanted to have a program to calculate square roots, or other roots, we would have to copy the code, possibly edit the variable names, include the extra code to do the square and other roots and paste it to where we want it
- The more code a program contains, the bigger the chance there is of an error occurring and the harder it is to maintain the program
- For example, say that we discovered that our cube root program had an error
- After fixing the error, we would then have to repeat that fix in the square root program and the general roots program

Functions in Python

- We have already used a number of built-in functions:
  - `range()` and `xrange()` `(in Python 2)`
  - `abs()`
  - `print()`
  - `...`
- The facility for programmers to define and subsequently use their own functions marks “a qualitative leap forward in convenience”

Defining functions in Python

- In Python, each function definition is of the following form:

  def *name of function* **(***list of formal parameters***)**:

  statement(s)

- **def** is a reserved keyword that introduces the definition of the function
- The function name is simply an identifier (a name) that is used to refer to the function

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

Formal and actual parameters

- The formal parameters of the function are the sequence of names within the parentheses after the function name (`(a, b)` in our example)
- When the function is used, ie at function invocation or function call, the formal parameters are bound to the values of the actual parameters or arguments
- This binding is similar to the binding that takes place in an assignment statement
- For example, the function call `max(2, 5)` binds `a` to `2` and `b` to `5`

Function body

- The function body is any piece of Python code
- This code, when executed, carries out the actions of the function
- A function invocation is an expression and, like all expressions, it has a value
- That value is the value returned by the invoked function
- There is a special statement, `return`, that can only be used within the body of a function
- This statement returns the value from the function to the expression in which it was invoked

The `return` statement

- For example, the value of the expression `max(2, 5)` is 5
- The value of the expression `max(4, 3) * max(2, 5)` is 20 (the first invocation of `max` returns 4 and the second invocation returns 5)
- Execution of a `return` statement terminates that invocation of the function
- If there is no value specified after the `return` statement, the special value `None` is returned
- If there is no `return` statement, the invocation of the function continues until there are no more statements to execute
- In this case, the value `None` is returned

Summary of function invocation

1. The expressions that make up the actual parameters are evaluated and the formal parameters of the function are bound to the resulting values
2. The point of execution (the next instruction to be executed) is moved from the point of invocation to the first statement in the body of the function
3. The code in the body of the function is executed until a `return` statement is executed, in which case the value of the expression following the `return` becomes the value of the function invocation, or there are no more statements to execute, in which case the function returns the value `None`
4. The value of the invocation of the function is the returned value
5. The point of execution is transferred back to the code immediately following the function invocation