

COMP1730/COMP6730 Programming for Scientists

Functions, part 2



Lecture outline

* Recap of functions and positional arguments

- keyword arguments
- * default arguments

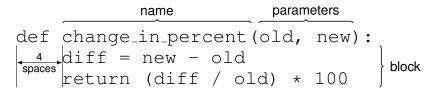


Functions (recap)

- * A *function* is a piece of code that can be *called* by its name.
- * Why use functions?
 - Abstraction: To use a function, we only need to know *what* it does, *not how*.
 - Readability.
 - Divide and conquer break a complex problem into simpler problems.
 - A function is a logical unit of testing.
 - Reuse: Write once, use many times (and by many).



Function definition



- * The function body is defined by indentation.
- Function *parameters* are variables local to the function body; their values are set when the function is called.
- The def statement only *defines* the function
 it does not execute the function.



Function call with positional arguments

 To call a function, write its name followed by its arguments in parentheses (this style of function calling is using *positional arguments*):

change_in_percent(485, 523)

- Order of evaluation: The argument expressions are evaluated left-to-right, and their values are assigned to the parameters; then the function body is executed.
- return *expression* causes the function call to end, and return the value of the expression.



Function call with keyword arguments

- In python, it is also possible to specify arguments by the formal parameter names, and give them in arbitrary order (this style of function calling is using *positional arguments*):
 - def change_in_percent(lastyear,
 thisyear): ...

change_in_percent(thisyear = 523, lastyear = 485)

 ★ If the function parameters are well named, this can make the argument call self-documenting.



Function default arguments

 Trailing arguments can be given default values that are used if no value is given at the function call site. (These are called *default arguments*):

def change_in_percent(lastyear,
thisyear=100.0): ...

change_in_percent(lastyear = 485)

- Library functions often have many arguments with defaults so that common usage is easy, but the function call can be customised if needed.
- Be careful with mutable default arguments, as they are assigned by reference, which can lead to unexpected results.



Function call return value

- * A function call is an expression: its value is the value return'd by the function.
- * In python, functions always return a value: If execution reaches the end of a function body without executing a return statement, the return value is the special value None of type NoneType.
- Note: None-values are not printed in the interactive shell (unless explicitly with print).



Guidelines for good functions

- * Within a function, access only local variables.
 - Use parameters for all inputs to the function.
 - Return all function outputs (for multiple outputs, return a tuple or list).
 - ...except if the *specific purpose* of the function is to send output elsewhere (e.g., print).
- Don't modify mutable argument values, unless the *specific purpose* of the function is to do that (and in that case document that the argument is modified by the function).
- * Rule #4: No rule should be followed off a cliff.