

COMP1730/COMP6730

Programming for Scientists

Functions, part 2

Announcements

- * Homework 4 is due at 9:00am next Monday
- * Homework 4 marking in the labs next week.
- * If you want to change lab groups next week, the deadline to do so is 12:00pm Monday.



Lecture outline

- * Recap of functions.
- * Namespaces & references.



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 call

- * To call a function, write its name followed by its *arguments* in parentheses:

```
change_in_percent (435, 499)
```

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

Functions without return

- * 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 suite 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`).



Namespaces

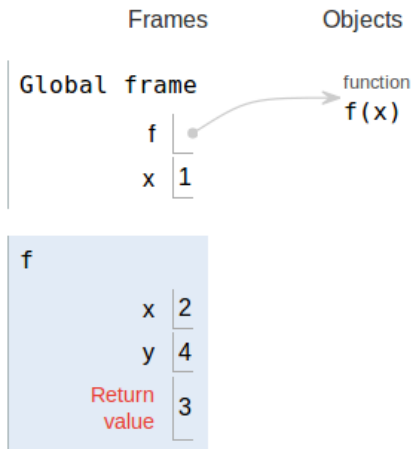
Namespaces

- * Assignment associates a (variable) name with a reference to a value.
 - This association is stored in a *namespace* (sometimes also called a “*frame*”).
- * Whenever a function is called, a new *local namespace* is created.
- * Assignments to variables (including parameters) during execution of the function are done in the local namespace.
- * The local namespace disappears when the function call ends.

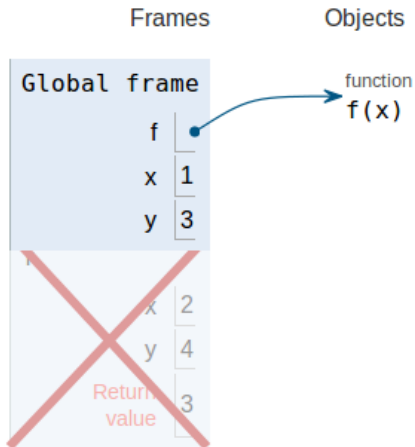
Scope

- * The *scope* of a variable is “the set of program statements over which a variable exists (i.e., can be referred to)”.
- In other words, the set of program statements over which the namespace that the variable is defined in persists.
- * Because there are several namespaces, there can be *different variables with the same name in different scopes*.

```
def f(x):  
    y = x ** 2  
    return y - 1  
  
x = 1  
y = f(x + 1)
```



```
def f(x):  
    y = x ** 2  
    return y - 1  
  
x = 1  
y = f(x + 1)
```



The local assignment rule

- * python considers a variable that is assigned **anywhere** in the function suite to be a “*local variable*” (this includes parameters).
- * When a non-local variable is evaluated, its value is taken from the (enclosing) global namespace.
- * When a local variable is evaluated, only the local namespace is checked.
 - If the variable is not defined there, python raises an `UnboundLocalError`.
- * The rule considers only *variable assignment*.



```
def f(x):  
    return x ** y  
  
>>> y = 2  
>>> f(2)  
4
```

```
def f(x):  
    if y < 1:  
        y = 1  
    return x ** y  
  
>>> y = 2  
>>> f(2)
```

UnboundLocalError:
local variable 'y'
referenced before
assignment



- * Modifying is not assignment!
 - Assignment changes/creates the association between a name and a reference (in the current namespace).
 - A modifying operation on a mutable object – including index and slice assignment – does not change any name–value association.

```
def f(x):  
    y = x ** 2  
    f_list.append([x,y])  
    return y  
  
>>> f_list = []  
>>> f(2)  
4  
>>> f(3)  
9  
>>> f_list  
[[2, 4], [3, 9]]
```


Argument values are references

- * When a function is called, its parameters are assigned *references* to the argument values.
 - If an argument value refers to a mutable object (for example, a list), modifications to this object made in the function are visible outside the function's scope.

```
def f(ns):  
    total = 0  
    while len(ns) > 0:  
        next = ns.pop(0)  
        total = total + next  
    return total  
  
>>> a_list = [1,2,3]  
>>> f(a_list)  
6  
>>> a_list  
[]
```

Frames

Objects

Global frame

f
a_listfunction
f(ns)

list

0	1	2
1	2	3

```
def f(ns):  
    total = 0  
    while len(ns) > 0:  
        next = ns.pop(0)  
        total = total + next  
    return total
```

```
>>> a_list = [1,2,3]  
>>> l_sum = f(a_list)
```

Image from pythontutor.com

Other namespaces

- * python's built-in functions are defined in a separate namespace.
- * Imported modules are executed in their own namespace.
 - Names in a module namespace are accessed by prefixing the name of the module.
- * User-defined classes and objects (not covered in this course) also have their own namespace
- * Assignments (and `defs`) made outside a function call are stored in the *global* namespace.

Searching for variables

- * When evaluating a variable python checks namespaces in a specific order LEGB.
 - Local - python checks in the local namespace (i.e. within the function definition).
 - Enclosing - within a class definition or an enclosing function definition.
 - Global - within the global namespace.
 - Built-ins - anything built into python.
- * Python uses the first version of the variable it finds.
- * If none of the namespaces contain the variable, python raises a `NameError`.

Why have namespaces?

- * Why have namespaces at all, why not just have everything global?
- * Namespaces are about organisation and access control.
- * Like most aspects of code quality, they become more important the larger the project.
- * Place limitations on the life of a variable and where it can be changed.
- * If anyone can modify any variable from anywhere in the project, and your project contains 2 million lines of code, how can you tell where (and why) a value was changed?

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.
- * **Rule #4:** No rule should be followed off a cliff.