functions_for_dummies

January 23, 2018

1 Functions

- What is a function?
- Why use functions?
- How to define and call functions?
- Keyword Arguments
- Default Argument Values
- Functions vs Methods
- Arbitrary Argument Lists

1.1 What is a function?

A function is a **block of organized, reusable code** that is used to **perform a single, specific task**. It performs actions on input data (arguments) and returns a result if necessary.

1.2 Why use functions?

- **Reusability**: functions are reusable blocks of code and allow to avoid copy/paste which is a dangerous practice and pollutes the code.
- **Organization**: functions help to organize your model. As a model grows in complexity, having all the code live inside a "main" script becomes increasingly complicated. Using functions allows to divide complicated tasks into smaller, simpler ones, and reduce the overall complexity of your model.
- **Abstraction**: Functions can be used as "black boxes", you don't need to know what is inside to use them. To use a function, you just need to know its name (= what it is supposed to do), the input arguments and the optional output.

1.3 How to define and call functions?

1.3.1 How to define functions?

- Function blocks begin with the keyword **def** followed by the **function name**, parentheses **( )** and a colon **:**
- Input **arguments**, if any, are placed within these parentheses.
- The code block within every function is indented.
- A function can optionally start by its **documentation**: a string written between triple quotes """ (multiple lines documentation is allowed).

- The optional statement **return [variable(s) or expression]** exits a function and returns output result(s).

Syntax:

```
def function_name(arguments):
    """function documentation = docstring (optional)"""
    <function code>
    # optional
    return [variable(s) or expression]
```

Let's begin with a simple function with no input arguments and returning nothing:

```
In [1]: def it_helpdesk():
            """Print universal solution to any computer problem"""
            print("Have you tried to turn it off and on again?")
```

Function with an input argument:

```
In [2]: def it_helpdesk(problem):
            print("Thank you for your request for support concerning:")
            print(problem)
            print("Before we do anything, have you tried to turn it off and on again?")
```

Function returning something:

```
In [3]: # (case 1) function returning an expression
        def it_helpdesk():
            return "Have you tried to turn it off and on again?"

        # (case 2) function returning a variable
        def it_helpdesk():
            answer = "Have you tried to turn it off and on again?"
            return answer
```

Function with an input argument and returning something:

```
In [4]: def it_helpdesk(problem):
            answer = "Thank you for your request for support concerning:\n"
            answer = answer + problem + "\n"
            answer = answer + "Before we do anything, have you tried to turn it off and on again
            return answer
```

1.3.2 How to call functions?

- To call a function, simply type its name followed by parentheses ().
- If the function requires input arguments, you need to provide values for them inside the parentheses. Theses values can be constants or expressions.
- If the function returns a result, it can be stored in a variable by preceding the function name with a variable name.

Syntax:

```
# function with no input argument and returning nothing
function_name()
# function with input arguments and returning nothing
function_name(arguments)
# function with input arguments and returning a result
res = function_name(arguments)
```

Function with no input arguments:

```
In [5]: def it_helpdesk():
            print("Have you tried to turn it off and on again?")

        # call and execute function "it_helpdesk()"
        it_helpdesk()
```

```
Have you tried to turn it off and on again?
```

Function with an input argument:

```
In [6]: def it_helpdesk(problem):
            print("Thank you for your request for support concerning:")
            print(problem)
            print("Before we do anything, have you tried to turn it off and on again?")

        # call function "it_helpdesk" and pass a string as input argument
        it_helpdesk("My computer smells weird and is very hot")

        print("\n10 minutes later...\n")

        # call function "it_helpdesk" and a variable
        # (!) the name of the passed variable can be different from the name of the input argume
        my_problem = "My computer is on fire!"
        it_helpdesk(my_problem)
```

```
Thank you for your request for support concerning:
My computer smells weird and is very hot
Before we do anything, have you tried to turn it off and on again?

10 minutes later...

Thank you for your request for support concerning:
My computer is on fire!
Before we do anything, have you tried to turn it off and on again?
```

Function returning something:

```
In [7]: def it_helpdesk():
            return "Have you tried to turn it off and on again?"

        # call function "it_helpdesk"
        it_answer = it_helpdesk()
        print(it_answer)
```

```
Have you tried to turn it off and on again?
```

Function with an input argument and returning something:

```
In [8]: def it_helpdesk(problem):
            answer =  "Thank you for your request for support concerning:\n"
            answer = answer + problem + "\n"
            answer = answer + "Before we do anything, have you tried to turn it off and on again
            return answer

        # call function "it_helpdesk" and pass a string as input argument
        it_answer = it_helpdesk("My computer smells weird and is very hot")
        print(it_answer)

        print("\n10 minutes later...\n")

        # call function "it_helpdesk" and a variable
        # (!) the name of the passed variable can be different from the name of the input argume
        user_problem = "My computer is on fire!"
        it_answer = it_helpdesk(user_problem)
        print(it_answer)
```

```
Thank you for your request for support concerning:
My computer smells weird and is very hot
Before we do anything, have you tried to turn it off and on again?

10 minutes later...

Thank you for your request for support concerning:
My computer is on fire!
Before we do anything, have you tried to turn it off and on again?
```

Note: a function can return several values:

```
In [9]: # Note: this function already exists in Python
        def divmod(a, b):
            return a // b, a % b

        quotient, remainder = divmod(5, 2)
        print(quotient, remainder)
```

```
2 1
```

1.4 Keyword Arguments

It is possible to pass input arguments to a function using the syntax **keyword=value**:

```
In [10]: def can_i_trust_this_website(website, country):
             if country == 'US' and 'foxnews' in website:
                 print("Yes")
             else:
                 print("No! It's all fake news!")

         # call function can_i_trust_this_website using "keywords arguments"
         can_i_trust_this_website(website="http://www.plan.be", country="Belgium")
```

```
No! It's all fake news!
```

In that case, arguments can be passed in any order:

```
In [11]: # call function can_i_trust_this_website and pass input arguments in reverse order
         can_i_trust_this_website(country="Belgium", website="http://www.plan.be")
```

```
No! It's all fake news!
```

It is even possible to mix positional and keyword arguments:

```
In [12]: can_i_trust_this_website("http://www.plan.be", country="Belgium")
```

```
No! It's all fake news!
```

WARNING: positional arguments must always be passed first

```
In [13]: can_i_trust_this_website(website="http://www.plan.be", "Belgium")
```

```
          File "<ipython-input-13-ea4a27a80b58>", line 1
        can_i_trust_this_website(website="http://www.plan.be", "Belgium")
                                                               ^
    SyntaxError: positional argument follows keyword argument
```

1.5 Default Argument Values

It is possible to set a default value to some arguments of a function:

```
In [14]: def can_i_trust_this_website(website, country='Belgium'):
             if country == 'US' and 'foxnews' in website:
                 print("Yes")
             else:
                 print("No! It's all fake news!")

         # if no value is passed for the "country" argument, it will be set to "Belgium" by defa
         can_i_trust_this_website("http://www.plan.be")
```

```
No! It's all fake news!
```

WARNING: arguments with default values must always be declared after all the others:

```
In [15]: # Wrong function definition --> arguments with default values must always be placed at
         def can_i_trust_this_website(website="http://www.foxnews.com", country):
             if country == 'US' and 'foxnews' in website:
                 print("Yes")
             else:
                 print("No! It's all fake news!")

         can_i_trust_this_website("Belgium")
```

```
  File "<ipython-input-15-85bf7bff3567>", line 2
    def can_i_trust_this_website(website="http://www.foxnews.com", country):
                                 ^
SyntaxError: non-default argument follows default argument
```

What about default value of **mutable** arguments like list, dictionary or array?

```
In [16]: # Wrong function defintion --> default value for mutable input arguments should be None
         # initialize inside the function
         def new_list_wrong_way(value, new_list=[]):
             new_list.append(value)
             return new_list

         result_1 = new_list_wrong_way(1)
         print('Expected [1]. Got:', result_1)

         result_2 = new_list_wrong_way(2)
         print('Expected [1]. Got:', result_2)

         result_3 = new_list_wrong_way(3)
         print('Expected [1]. Got:', result_3)
```

```
Expected [1]. Got: [1]
Expected [1]. Got: [1, 2]
Expected [1]. Got: [1, 2, 3]
```

The default value is evaluated only once. This leads to an unexpected behavior when the default value is an object of composed type such as a list or dictionary.

WARNING: To define a function with arguments having a list, a dictionary or an array as default value, use **None** as the default value in the function definition and and **set the default value at the beginning of the function**:

```
In [17]: # Right way
         def new_list_right_way(a, new_list=None):
             if new_list is None:
                 new_list = []
             new_list.append(a)
             return new_list

         result_1 = new_list_right_way(1)
         print('Expected [1]. Got:', result_1)

         result_2 = new_list_right_way(2)
         print('Expected [1]. Got:', result_2)

         result_3 = new_list_right_way(3)
         print('Expected [1]. Got:', result_3)
```

```
Expected [1]. Got: [1]
Expected [1]. Got: [2]
Expected [1]. Got: [3]
```

Note: Keywords arguments and default argument values are two different things: - **Keywords arguments**: "name=value" in **function call** - **Default argument values**: "name=value" in **function definition**

An argument without a default value can be passed as keyword argument and an argument with default value can be used like any positional argument:

```
In [18]: def function_with_default_argument_value(positional_arg, arg_with_default_value="argume
             print(positional_arg)
             print(arg_with_default_value)
             print()

         # An argument without a default value can be passed as keyword argument
         function_with_default_argument_value(positional_arg="positional arg passed as keyword a

         # An argument with default value can be used like any positional argument
         function_with_default_argument_value("positional arg", "override default value")
```

```
positional arg passed as keyword argument
argument with a default value

positional arg
override default value
```

1.6 Functions vs Methods

A method is a function called on a object using the syntax `object.method(arguments)`:

```
In [19]: from larray import ones

         pop = ones('age=0..5')
         print(pop)

         # call method 'sum' on object 'pop'
         total_pop = pop.sum('age')
         print()
         print("total population:", total_pop)
```

```
age    0    1    2    3    4    5
     1.0  1.0  1.0  1.0  1.0  1.0

total population: 6.0
```

1.7 Arbitrary Argument Lists

Some functions or methods have special input arguments `*args` and `**kwargs`.

A function which has such arguments can have an arbitrary number of arguments: - `*args` for positional arguments (arguments passed without keyword)

```
In [20]: def function_with_arbitrary_positional_arguments(*args):
             # passed arguments are converted to a tuple
             print(args)

         function_with_arbitrary_positional_arguments(0, 1, 2, 3, 4)
```

```
(0, 1, 2, 3, 4)
```

- `**kwargs` for keyword arguments

```
In [21]: def function_with_arbitrary_keyword_arguments(**kwargs):
             # passed arguments are converted to a dictionary
             print(kwargs)

         function_with_arbitrary_keyword_arguments(firstname='Sarah', name='Connor', country='US
```

```
{'firstname': 'Sarah', 'name': 'Connor', 'country': 'US'}
```

The builder method for Session is a good example:

```
In [22]: from larray import Axis, Session, ones

         # define axes
         AGE = Axis('age = 0..5')
         GENDER = Axis('gender = F,M')
         COUNTRY = Axis('country = BE,FR,IT,UK')
         # define arrays
         pop_be = ones((AGE, GENDER))
         pop_by_age = ones((AGE, COUNTRY))
         pop_all = ones((AGE, GENDER, COUNTRY))

         # store axes and arrays in a session
         # Session builder accepts an arbitrary number of axes and arrays.
         # Axes are passed first and separated with commas (*args).
         # Arrays are then passed as keyword arguments (**kwargs).
         ses = Session(AGE, GENDER, COUNTRY, pop_be=pop_be, pop_by_age=pop_by_age, pop_all=pop_a

         print(ses)

Session(age, gender, country, pop_be, pop_by_age, pop_all)
```

1.8 More infos on defining functions?

See the official documentation of Python (3.5)