3.4 - Functions

Ha Khanh Nguyen (hknguyen)

1. What is a Function?

- Function here means user-defined function.
- If you anticipate needing to repeat the same or very similar code more than once, it may be worth writing a reusable function.

```
# define the function
def my_function(x, y, z=1.5):
  if z > 1:
    return z * (x + y)
  else:
    return z / (x + y)
```

```
# execute the function
my_function(1, 2, 3)
```

```
## 9
```

```
my_function(1, 2)
```

```
## 4.5
```

- Use `def` keyword to define a function.
- A function might have argument(s). Argument(s) is not required for a function.
  - In `my_function()` above, `x` , `y` , and `z` are the arguments.
  - `z=1.5` means that if no value is provided for `z` , the default value of `z` is 1.5.
- Here is an example of a function with no argument:

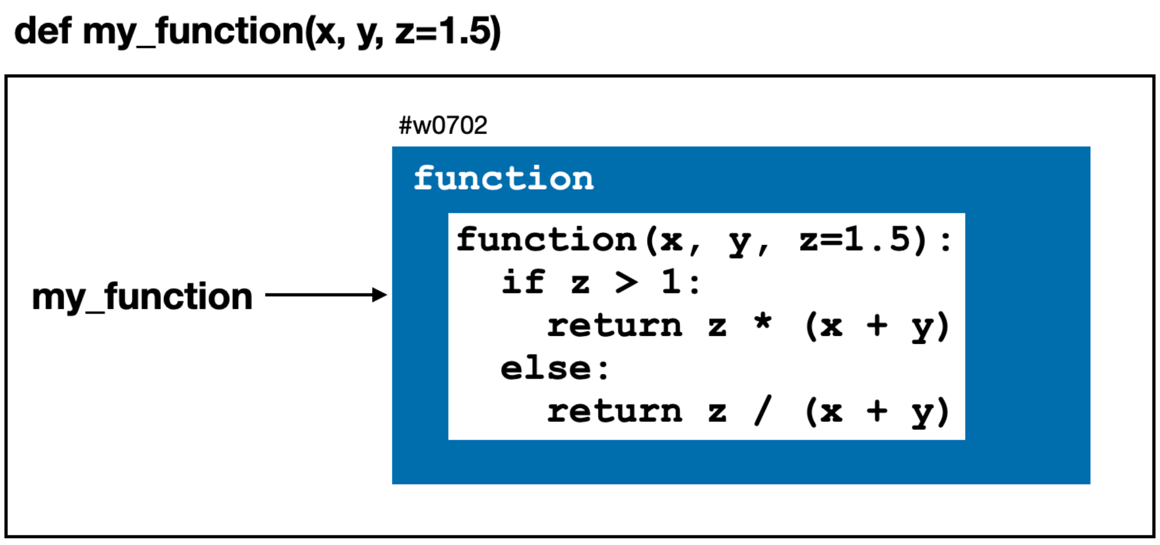
```
# define the function
def function_without_argument():
  print('I\'m still a function!')
```

```
# execute the function
function_without_argument()
```

```
## I'm still a function!
```

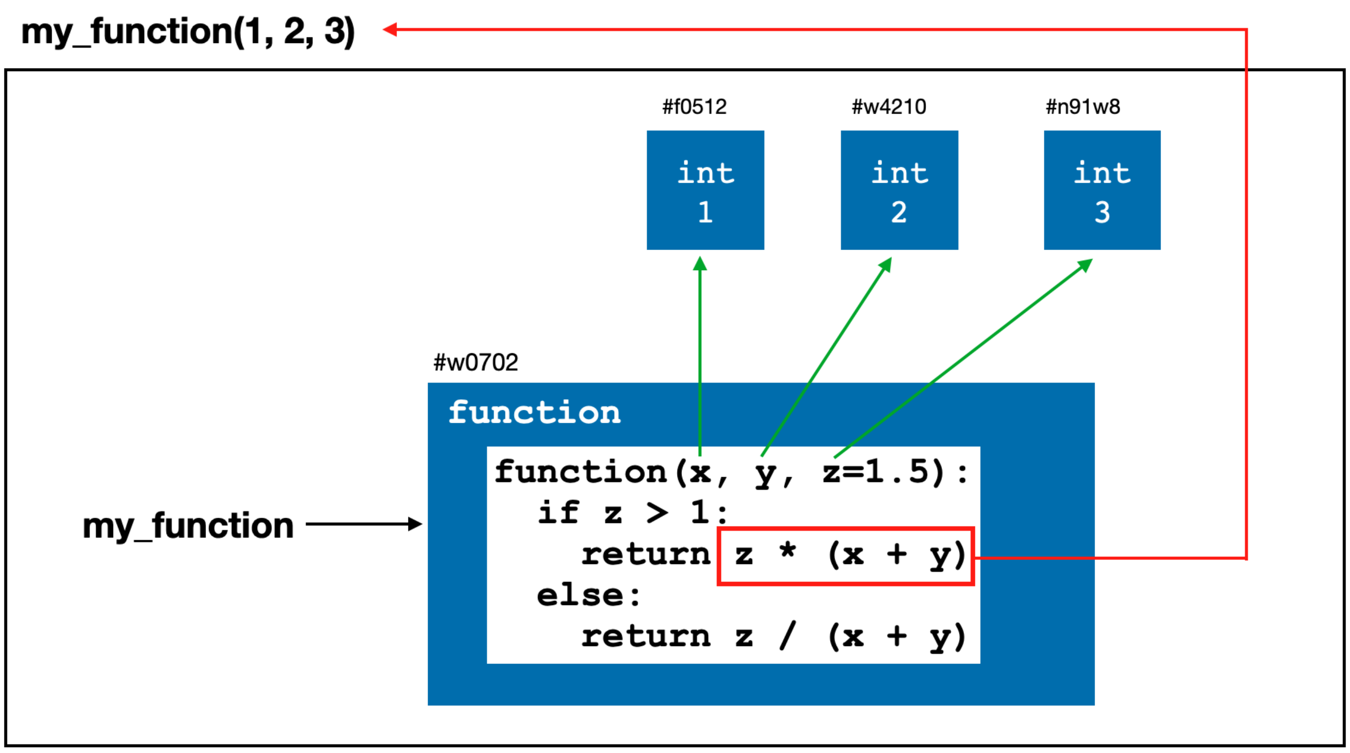
1.1 Defininig a function vs. a function call?

```
# define the function
def my_function(x, y, z=1.5):
  if z > 1:
    return z * (x + y)
  else:
    return z / (x + y)
```



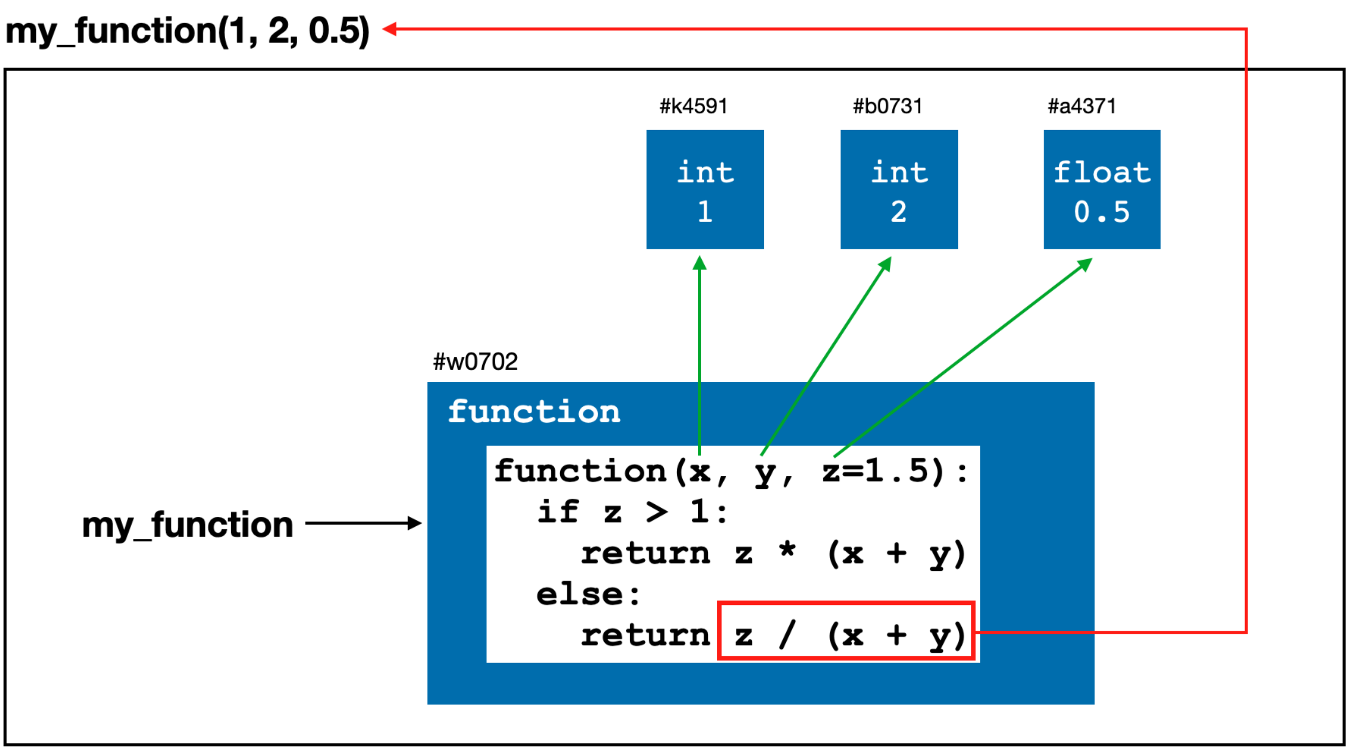
```
# execute the function
my_function(1, 2, 3)
```

```
## 9
```



```
# execute the function
my_function(1, 2, 0.5)
```

```
## 0.16666666666666666
```



2. Function Properties

2.1 Namespaces, scope, and local functions

- Functions can access variables in two different scopes: *global* and *local*.
- A variable scope in Python is also called a **namespace**.
- Any variables that are assigned **within** a function by default are assigned to the **local namespace**.
- The local namespace is created when the function is called and immediately populated by the function's arguments.
- After the function is finished, the local namespace is destroyed (with a few exceptions).

```
def func():
  a = []
  for i in range(5):
    a.append(i)
```

```
func()
print(a)
```

```
## Error in py_call_impl(callable, dots$args, dots$keywords): NameError: name 'a' is
not defined
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```

```
a=[]
def another_func():
  for i in range(5):
    a.append(i)
```

```
another_func()
print(a)
```

```
## [0, 1, 2, 3, 4]
```

2.2 Returning multiple values

```
def f():
  a = 5
  b = 6
  c = 7
  return a, b, c
```

```
a, b, c = f()
```

- The function is actually just returning one object, namely a tuple, which is then being unpacked into the result variables.
- We can also return a dictionary instead!

```
def f():
  a = 5
  b = 6
  c = 7
  return {'a': a, 'b': b, 'c': c}
```

```
return_value = f()
return_value['a']
```

```
## 5
```

2.3 Functions are objects

- Suppose we were doing some data cleaning and needed to apply a bunch of transformations to the following list of strings:

```
states = [' Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda', 'south carolina#
#', 'West virginia?']
```

Exercise

Write a function to clean a list of strings, the tasks include: stripping whitespace, removing punctuation symbols, and standardizing on proper capitalization.

- An alternative approach is to make a list of operations we want to apply to this set of strings!

```
def remove_punctuation(value):
  # to be completed
  pass

clean_ops = [str.strip, remove_punctuation, str.title]

def clean_strings(strings, ops):
  result = []
  for value in strings:
    for function in ops:
      value = function(value)
      result.append(value)
  return result
```

```
clean_strings(states, clean_ops)
```

2.4 Anonymous (lambda) functions

- Anonymous or lambda functions are ways of writing functions consisting of a single statement, the result of which is the return value.

```
def short_function(x):
  return x * 2

equiv_anon = lambda x: x * 2
```

```
def apply_to_list(some_list, f):
  return [f(x) for x in some_list]

ints = [4, 0, 1, 5, 6]
apply_to_list(ints, lambda x: x * 2)
```

```
## [8, 0, 2, 10, 12]
```

This lecture note is modified from Chapter 3 of Wes McKinney's Python for Data Analysis 2nd Ed (https://www.oreilly.com/library/view/python-for-data/9781491957653/).