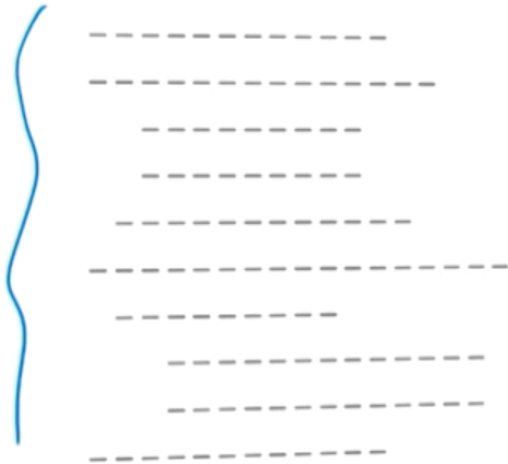
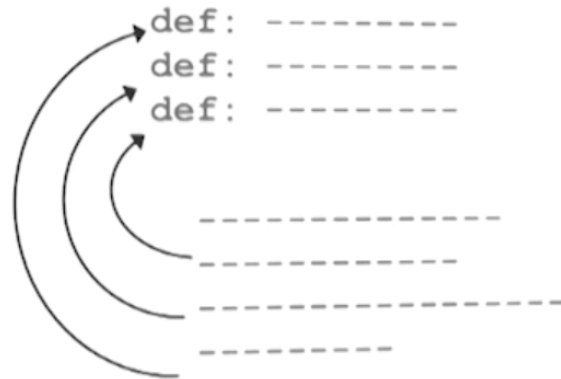


# Functions: Organizing Code

## Disorganized code



## Organized code

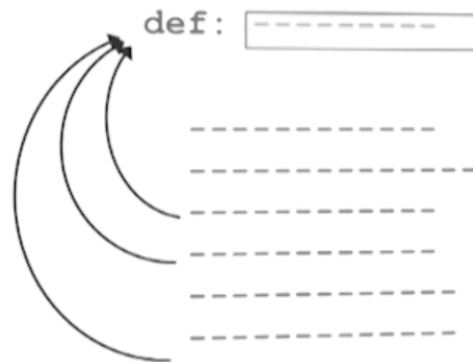


# Functions: Re-using Code

## Blocks of repeated code



## Define once, call many times



# Overview: Functions

- **Function Definition** ✓

Defining the code that will be called from other points in your application.

- **Function Calls** ✓

Calling the function you have defined, receiving results

- **Variable Scope** ✓

Where variables are valid, what they reference

- **Documenting Functions**

Describing the purpose and parameters of the function for others

## Parameters by Position

```
def connect(dev_IP, username, password):  
    ...  
    ...  
  
...  
session = connect('10.0.0.1', 'cisco', 'cisco')
```

must match positionally

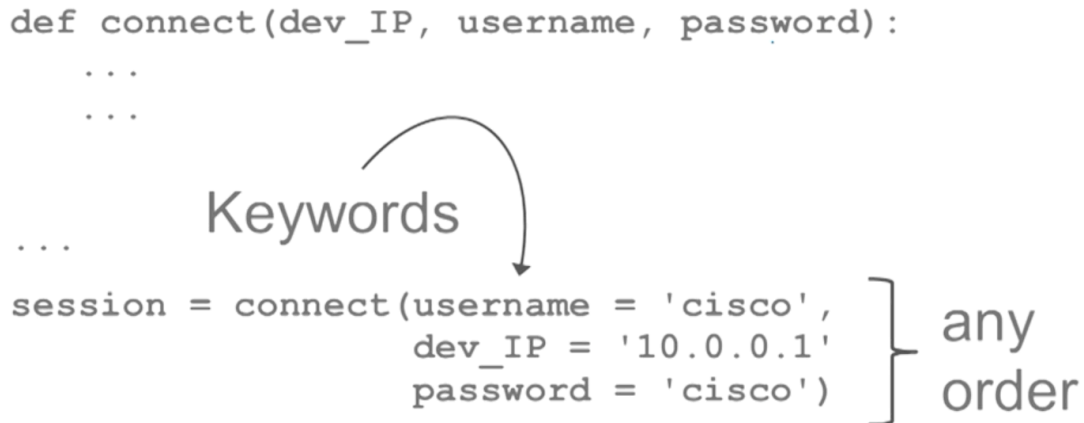
The diagram illustrates positional matching between function parameters and arguments. Three arrows point from the arguments '10.0.0.1', 'cisco', and 'cisco' in the function call to the parameters 'dev\_IP', 'username', and 'password' in the function definition, respectively. The text 'must match positionally' is placed between the function definition and the function call, with arrows pointing to the corresponding parameter and argument pairs.

# Parameters by Keyword

```
def connect(dev_IP, username, password):  
    ...  
    ...  
    ...  
session = connect(username = 'cisco',  
                  dev_IP = '10.0.0.1',  
                  password = 'cisco')
```

Keywords

} any order



# Function Return Values

```
def connect(dev_IP, username, password):  
    ...  
    return session
```

value to return to caller

```
...  
session = connect('10.0.0.1', 'cisco', 'cisco')
```

returned value



One of the most important facets of any programming language, including Python, is the ability to group certain pieces of code into functions, sometimes called methods, or subroutines in other languages.

Very long sequences of code become complicated, difficult to write, read, and debug. By taking certain parts of the code and organizing them into functions, each with a specific purpose, you improve your application during development time, and for readability once your code gets passed along to another set of developers.

Suitable blocks of functionality in a networking environment are:

- Connecting to a device
- Executing a command on a device
- Printing device information
- Searching for specific data from a device

Another advantage of using functions is to reduce, and hopefully eliminate, replicated code. If there is a specific operation that is invoked over and over in your application, it is a bad idea to repeat that code everywhere that it is needed.

A far better choice is to put that repeated code into a single function, which can be called from multiple places in your code.

- **Function Parameters:** What happens if the same code needs to be called, but with minor modifications to certain details? This is where the ability to pass arguments to your function is important. Using this mechanism, you can call the same code, but with varying input parameters, and have the code be reused for each instance, without code replication.
- **Function Return Values:** What happens if the code needs to create some resulting value that is needed by the calling code? This is where the ability to return values to the calling program is useful; values can be as simple as a number, a string, or a large data structure.

This lesson will examine:

- **Definition:** How to define a function.
- **Call:** How to call a function.
- **Scope:** How variable names work within a function
- **Documentation:** How to document your function.

```

#---- Function to read device information from file -----
def read_device_info():

    # Read in the devices from the file
    file = open('devices','r')
    for line in file:

        device_info_list = line.split(',') # Get device info into list
        devices_list.append(device_info_list)

    file.close() # Close the file since we are done with it

#---- Function to go through devices printing them to table -----
def print_device_info():

    print ''
    print 'Name      OS-type   IP address      Software      '
    print '-----  -----  -----  -----'

    # Go through the list of devices, printing out values in nice format
    for device in devices_list:

        print '{0:8} {1:8} {2:20} {3:20}'.format(device[0],device[1],
                                                device[2],device[3])

#---- Main: read device info, then print -----
devices_list = [] # Create the outer list for all devices

read_device_info()
print_device_info()

```