API and Python training

Session 6

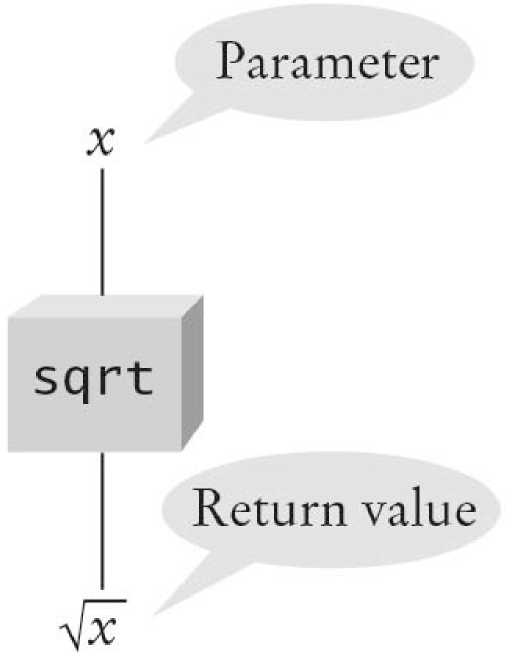
This session agenda

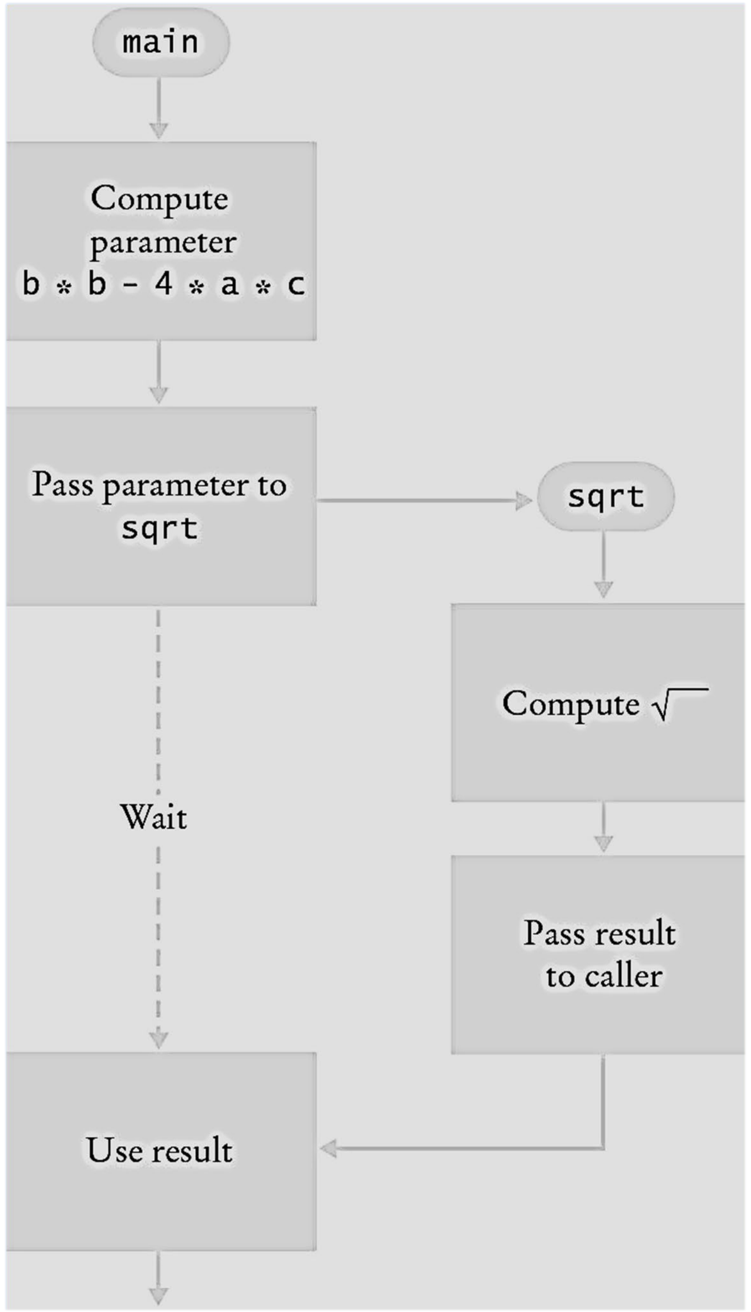
- Functions in programming languages
- Functions in Python
- Function arguments
- Position vs named arguments
- Variable scope
- Return values
- Demo

Functions in programming languages

- **Re-usable** block of code which you can run by calling it
- Can have **arguments** (also called parameters)
- Can **return values** – you can use returned value in the main program
- When you use the function name in the main program you **call** it

- The function is 'black box' – you supply parameters and get a result, you don't need to know the internal function's implementation details





https://www.ooportal.com/building-cplus-classes/module2/black-box-principle.php

Functions in Python

- Syntax:

  def <function_name> (parameters):

    <block of code>

- The block of code within the function has **indentation to** define where the function starts and ends
- Functions can be in the same module as your main program or in different – in this case **import** it

def function1(): <----------- defining functions

command1

command2

etc

def function2():

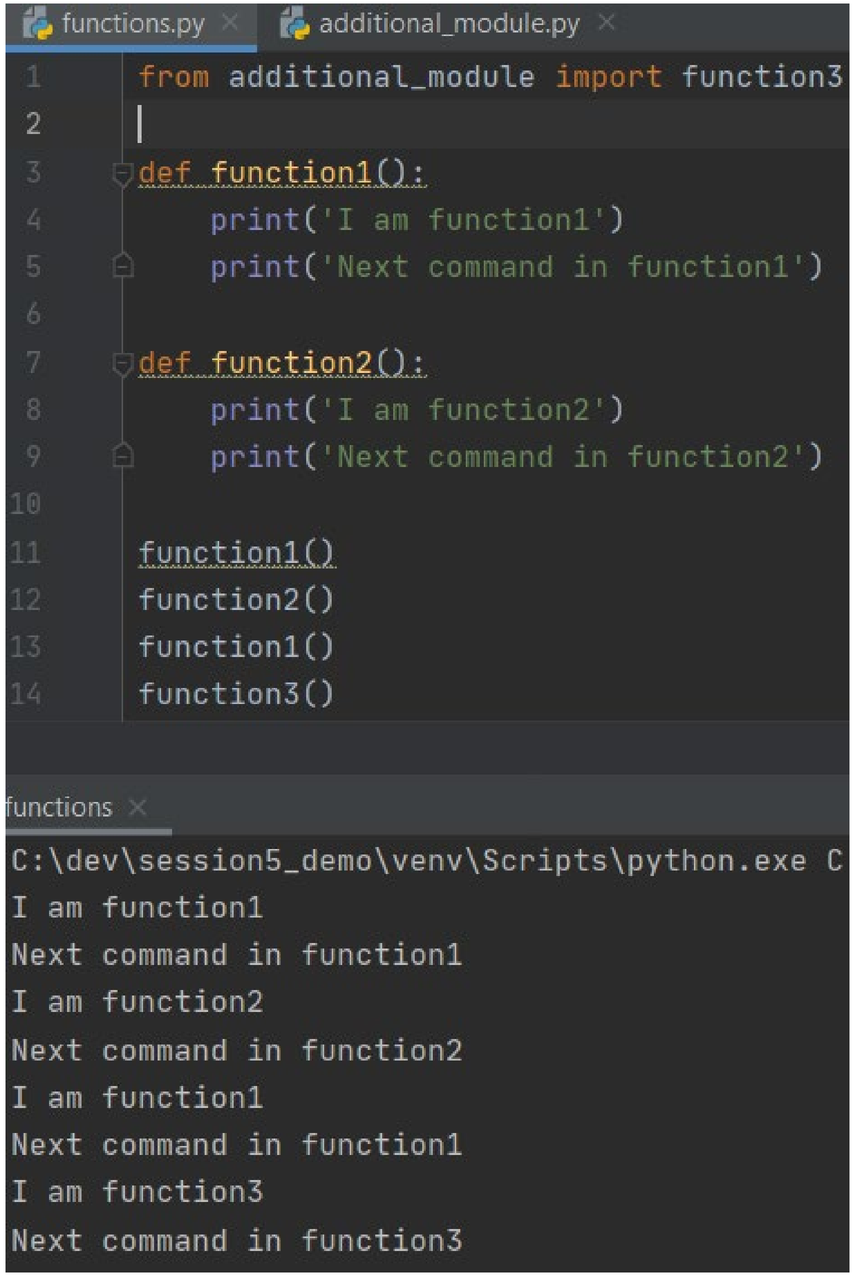
command1

command2

etc

function1() <--------- calling functions

function2()

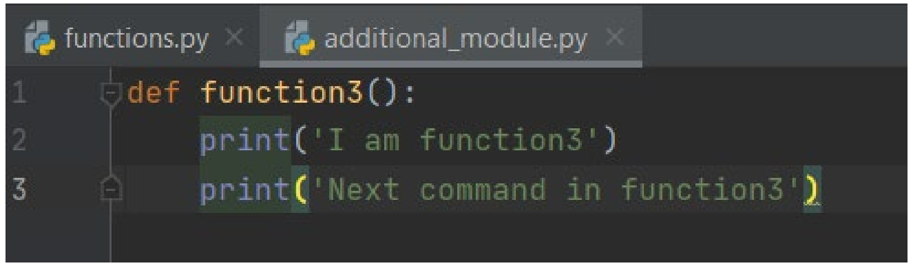


functions.py additional_module.py

```
from additional_module import function3

def function1():
    print('I am function1')
    print('Next command in function1')

def function2():
    print('I am function2')
    print('Next command in function2')

function1()
function2()
function1()
function3()
```

functions

```
C:\dev\session5_demo\venv\Scripts\python.exe C
I am function1
Next command in function1
I am function2
Next command in function2
I am function1
Next command in function1
I am function3
Next command in function3
```



functions.py additional_module.py

```
def function3():
    print('I am function3')
    print('Next command in function3')
```

https://www.tutorialspoint.com/python/python_functions.htm

Arguments

- When you use arguments/parameters you **pass** them to the function.

- A function may not have any arguments, so just use empty brackets:

  def <name>():

- **Example 1 - The number of arguments** you pass from the main program should be the same as defined in the function (there are some **special cases** though – default argument and arbitrary arguments)

  Correct:

  ```
  def my_func(my_param1, my_param2):
        print(my_param1)
  my_func(param1, param2)
  ```

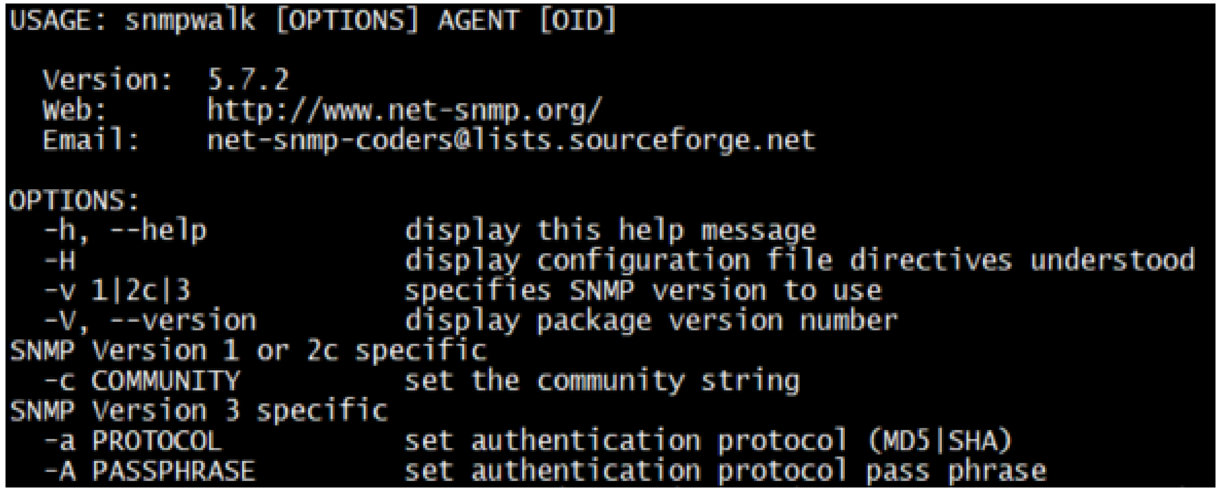
  Incorrect:

  ```
  def my_func(my_param1, my_param2):   <--- two arguments defined
        print(my_param1)
  my_func(param1)           <------ but called with only 1 argument
  ```

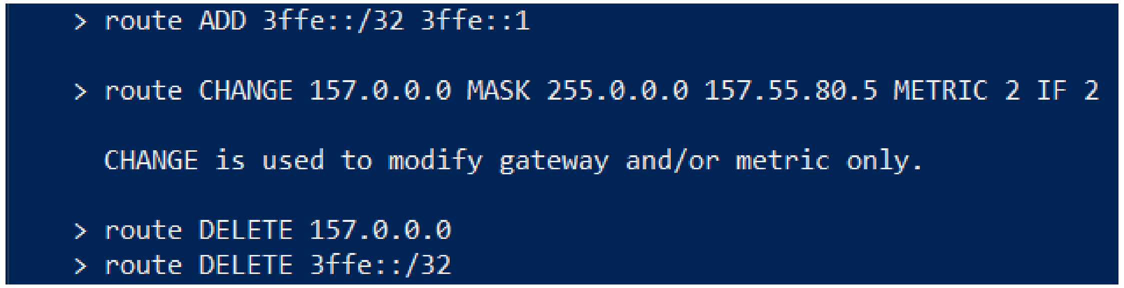
- **Example 2 - Special case - default arguments**

```
def my_func(my_param1, my_param2 = 'some default value'):
      print(my_param1)
my_func(param1)                   <------ when calling this function, you can omit default argument my_param2
```

Arguments – positioned vs named



```
USAGE: snmpwalk [OPTIONS] AGENT [OID]

  Version:  5.7.2
  Web:      http://www.net-snmp.org/
  Email:    net-snmp-coders@lists.sourceforge.net

OPTIONS:
  -h, --help            display this help message
  -H                    display configuration file directives understood
  -v 1|2c|3             specifies SNMP version to use
  -V, --version         display package version number
SNMP Version 1 or 2c specific
  -c COMMUNITY          set the community string
SNMP Version 3 specific
  -a PROTOCOL           set authentication protocol (MD5|SHA)
  -A PASSPHRASE         set authentication protocol pass phrase
```



```
    > route ADD 3ffe::/32 3ffe::1

    > route CHANGE 157.0.0.0 MASK 255.0.0.0 157.55.80.5 METRIC 2 IF 2

      CHANGE is used to modify gateway and/or metric only.

    > route DELETE 157.0.0.0
    > route DELETE 3ffe::/32
```

^^^ here we specific parameters (network, mask) by their position

<- here we use parameter names (such as -c -v) in any order

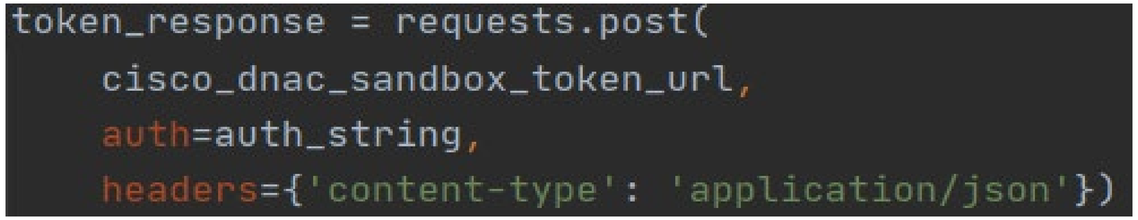
- **Positional arguments - the order of arguments** is important

```
def divide(a,b,c):
        print('dividing:', a/b/c)
divide(10,5,2)          --> result is 1
```

```
def divide(b,a,c):
        print('dividing:', a/b/c)
divide(10,5,2)          --> result is 0.25
```

- **Named arguments - the order of arguments** can be any, but you need to know variable names in function definition

```
def divide(a,c,b):
      print('dividing:', a/b/c)
divide(b=5, a=10, c=2)        --> result is 1
```



```
token_response = requests.post(
    cisco_dnac_sandbox_token_url,
    auth=auth_string,
    headers={'content-type': 'application/json'})
```

https://stackoverflow.com/questions/9450656/positional-argument-v-s-keyword-argument

Variable scope

- Variable names you use in the function definition will become **function variables** and available for use inside the function

def my_func1(url, token):

print(url, user) <------- user variable is not defined in this function, but url and token is

my_func1('https://api-server, 'abcd')

- Variables defined in functions have **function scope** – you can't access it outside the function

def my_func():

my_var = 'my awesome function variable'

print(my_var) <--------- **Incorrect,** my_var is defined in function, but this is **outside the function, note indentation**

- You can **re-use** the same the same variable name in different functions – they are different

def my_func1():

response_code = 200 \

def my_func2(): you can use **the same variable name** *response_code* in different functions

response_code = 404 /

Functions returning values

- Can **return values** – you can use returned value in the main program
- When you use the function name in the main program you **call** it

Example 1. No returned value:

```
    def get_sum (var1, var2, var3):
        sum = var1 + var3 +var2
        print(sum)

    get_sum(10,20,30)                <--- main program, we call this function
```

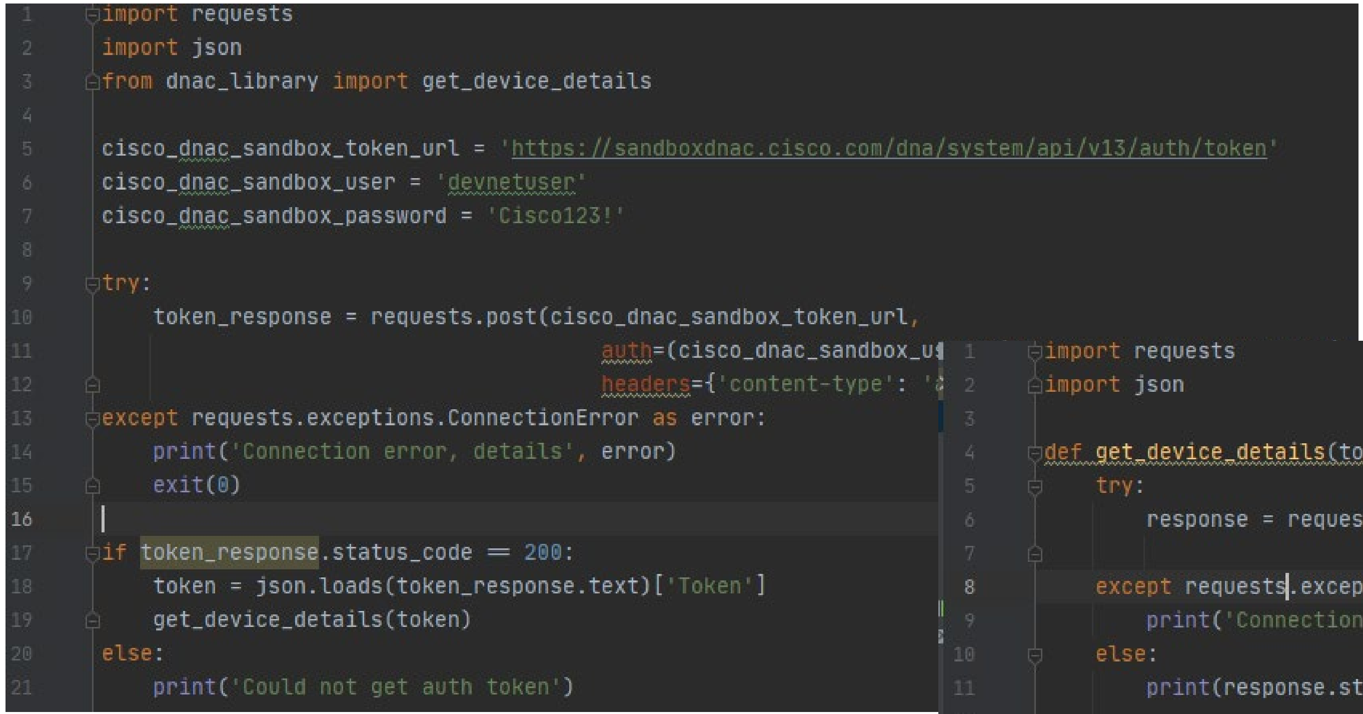
Example 2. With returned value:

```
 def get_sum (var1, var2, var3):
        sum = var1 + var2 +var3
        return sum              <---- 2. after the function completes, this will be the value the caller use in place of the function call

print(get_sum(10, 20, 30))     <---  1. we call the function from the main program, pass arguments 10,20,30 and print 60
```

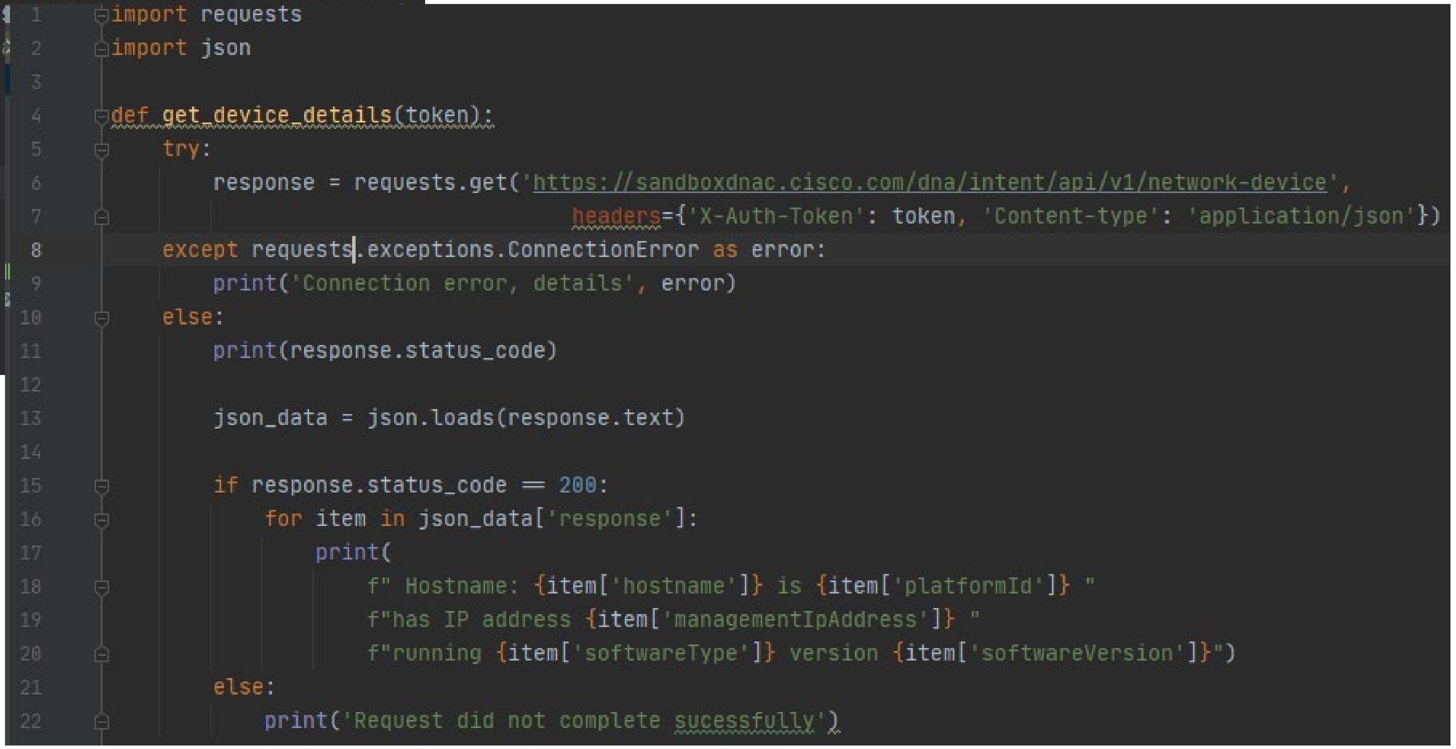
https://www.ooportal.com/building-cplus-classes/module2/black-box-principle.php

Demo

Demo 5 – defining functions, using modules



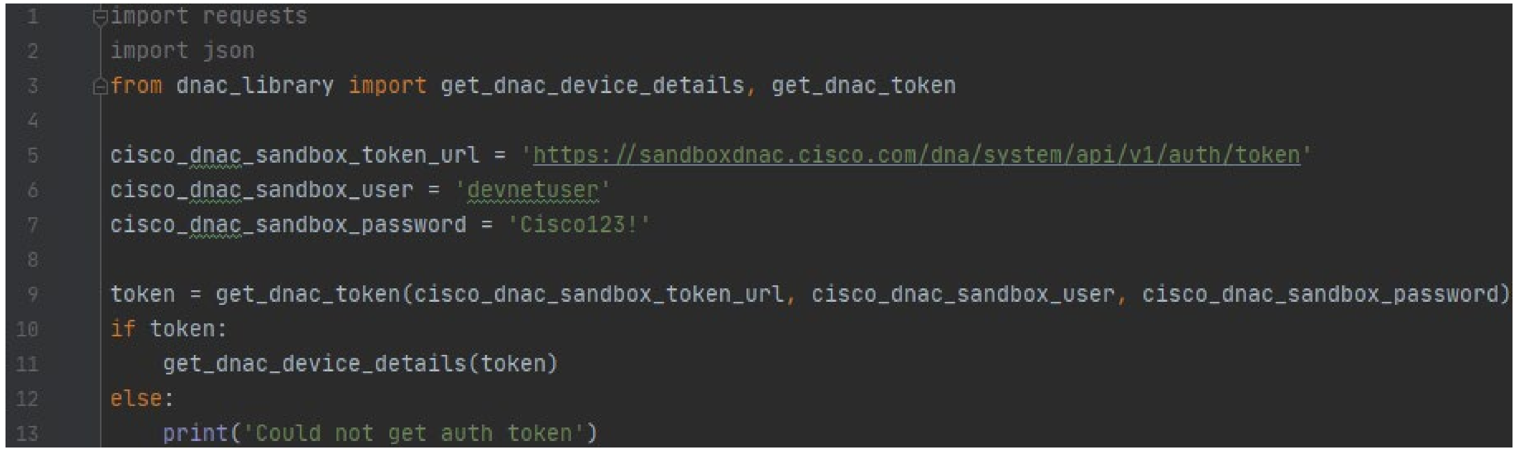
```
1   import requests
2   import json
3   from dnac_library import get_device_details
4
5   cisco_dnac_sandbox_token_url = 'https://sandboxdnac.cisco.com/dna/system/api/v13/auth/token'
6   cisco_dnac_sandbox_user = 'devnetuser'
7   cisco_dnac_sandbox_password = 'Cisco123!'
8
9   try:
10      token_response = requests.post(cisco_dnac_sandbox_token_url,
11                                     auth=(cisco_dnac_sandbox_us
12                                     headers={'content-type': '
13  except requests.exceptions.ConnectionError as error:
14      print('Connection error, details', error)
15      exit(0)
16
17  if token_response.status_code == 200:
18      token = json.loads(token_response.text)['Token']
19      get_device_details(token)
20  else:
21      print('Could not get auth token')
```



```
1   import requests
2   import json
3
4   def get_device_details(token):
5       try:
6           response = requests.get('https://sandboxdnac.cisco.com/dna/intent/api/v1/network-device',
7                                   headers={'X-Auth-Token': token, 'Content-type': 'application/json'})
8       except requests.exceptions.ConnectionError as error:
9           print('Connection error, details', error)
10      else:
11          print(response.status_code)
12
13          json_data = json.loads(response.text)
14
15          if response.status_code == 200:
16              for item in json_data['response']:
17                  print(
18                      f" Hostname: {item['hostname']} is {item['platformId']} "
19                      f"has IP address {item['managementIpAddress']} "
20                      f"running {item['softwareType']} version {item['softwareVersion']}")
21          else:
22              print('Request did not complete sucessfully')
```

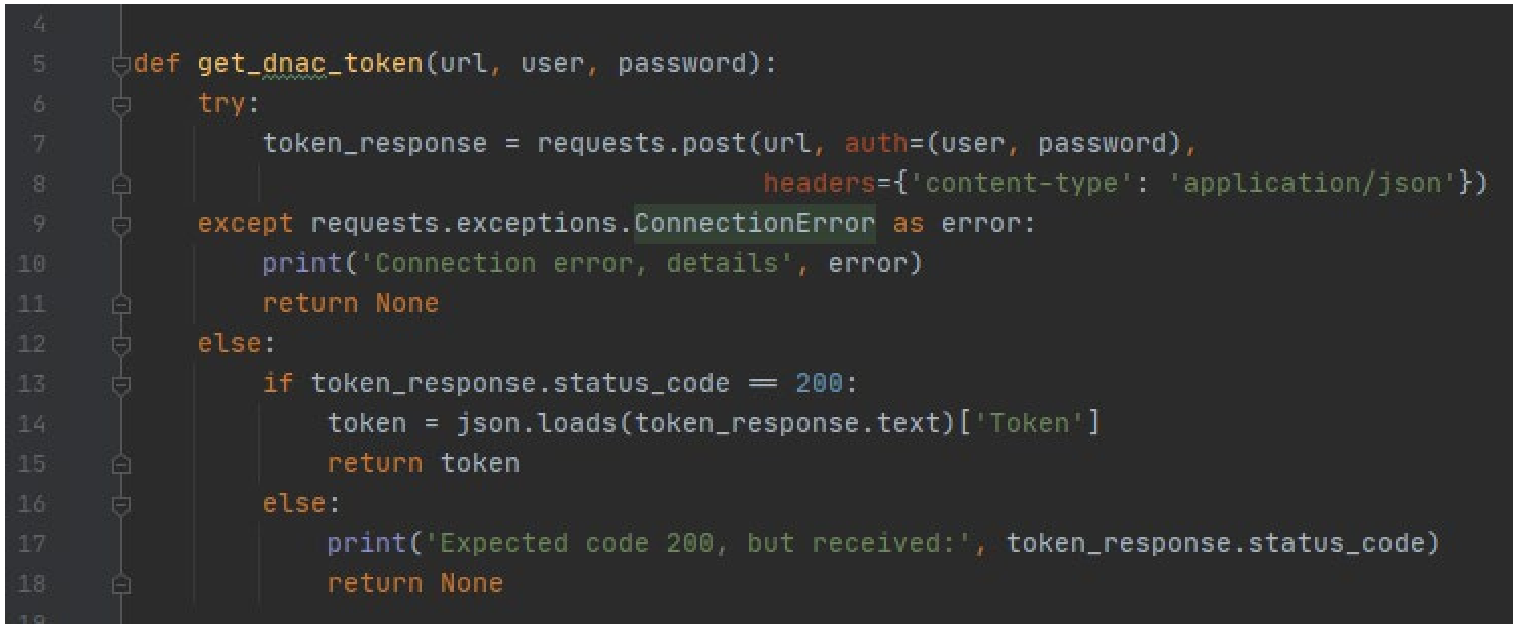
- <--- Main program, note Line 3 – import and Line 19 – calling function *get_device_details* with argument *token*
- Module is below, Line 4 – definition of function

Demo 6 – defining functions with return values



```python
1   import requests
2   import json
3   from dnac_library import get_dnac_device_details, get_dnac_token
4
5   cisco_dnac_sandbox_token_url = 'https://sandboxdnac.cisco.com/dna/system/api/v1/auth/token'
6   cisco_dnac_sandbox_user = 'devnetuser'
7   cisco_dnac_sandbox_password = 'Cisco123!'
8
9   token = get_dnac_token(cisco_dnac_sandbox_token_url, cisco_dnac_sandbox_user, cisco_dnac_sandbox_password)
10  if token:
11      get_dnac_device_details(token)
12  else:
13      print('Could not get auth token')
```

- <--- Main program, note Line 1 and 2 – we don’t need import requests and json anymore
- Line 3 – import from functions from your module
- Below is the function defined in the module
- Note Line 15 – we return token only of there were no exceptions and code is OK, other wise we return empty value



```python
4
5   def get_dnac_token(url, user, password):
6       try:
7           token_response = requests.post(url, auth=(user, password),
8                                          headers={'content-type': 'application/json'})
9       except requests.exceptions.ConnectionError as error:
10          print('Connection error, details', error)
11          return None
12      else:
13          if token_response.status_code == 200:
14              token = json.loads(token_response.text)['Token']
15              return token
16          else:
17              print('Expected code 200, but received:', token_response.status_code)
18              return None
```

Summary and next steps

- **Summary**

Python – functions

Next time

- Classes and objects
- Using SDKs