

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS



ÁREA DE CIENCIA DE LA COMPUTACIÓN

3° LABORATORIO - CC312

1 PARTE

- **TÍTULO:** CREAR UNA PRUEBA UNITARIA DE PYTHON

- **ALUMNO:**

JHONATAN POMA MARTINEZ

20182729F

- **PROFESORES:** YURI JAVIER.,CCOICCA PACASI

2023

OBJETIVOS:

Parte 1: Iniciar la máquina virtual (Virtual Machine) de DEVASC.

Parte 2: Explorar las opciones en el unittest del framework.

Parte 3: Probar una función de Python con unittest.

INTRODUCCIÓN:

Los unittest examinan unidades independientes de código, como funciones, clases, módulos y librerías. Hay muchas razones para escribir un script usando la librería unittest de Python. Una razón obvia es que si encuentra un problema en el código aislado mediante pruebas deliberadas, sabrás que el problema está en la función u otra unidad bajo prueba. El problema no está en la aplicación más grande que pueda llamar a esta función. También sabrá exactamente qué desencadenó el error porque el unittest escrito por el usuario expone el problema. Los errores encontrados de esta manera suelen ser rápidos y fáciles de corregir, y las correcciones hechas en este nivel detallado tienen menos probabilidades de causar efectos secundarios imprevistos más adelante en otro código que se basa en el código probado.

- El unittest debe ser simple y fácil de implementar.
- El código de prueba debe funcionar independientemente del código que se esté probando. Si escribe pruebas que necesitan cambiar el estado del programa, capture el estado antes de cambiarlo, y vuelva a cambiarlo después de que se ejecute la prueba.
- Cuando una prueba falla, los resultados deben ser fáciles de leer y señalar claramente lo que se espera y dónde están los problemas.

PARTE 1: Iniciar la máquina virtual (Virtual Machine) de DEVASC.

PARTE 2: Explorar las opciones en el unittest del framework.

TestCase - Caso de prueba

Un caso de prueba es la unidad individual de prueba. Comprueba una respuesta específica a un conjunto particular de entradas. unittest proporciona una clase base, TestCase se puede usar para crear nuevos casos de prueba.

TestRunner - Corredor de Pruebas

Un ejecutor de pruebas es un componente que organiza la ejecución de pruebas y proporciona el resultado al usuario. El corredor puede usar una

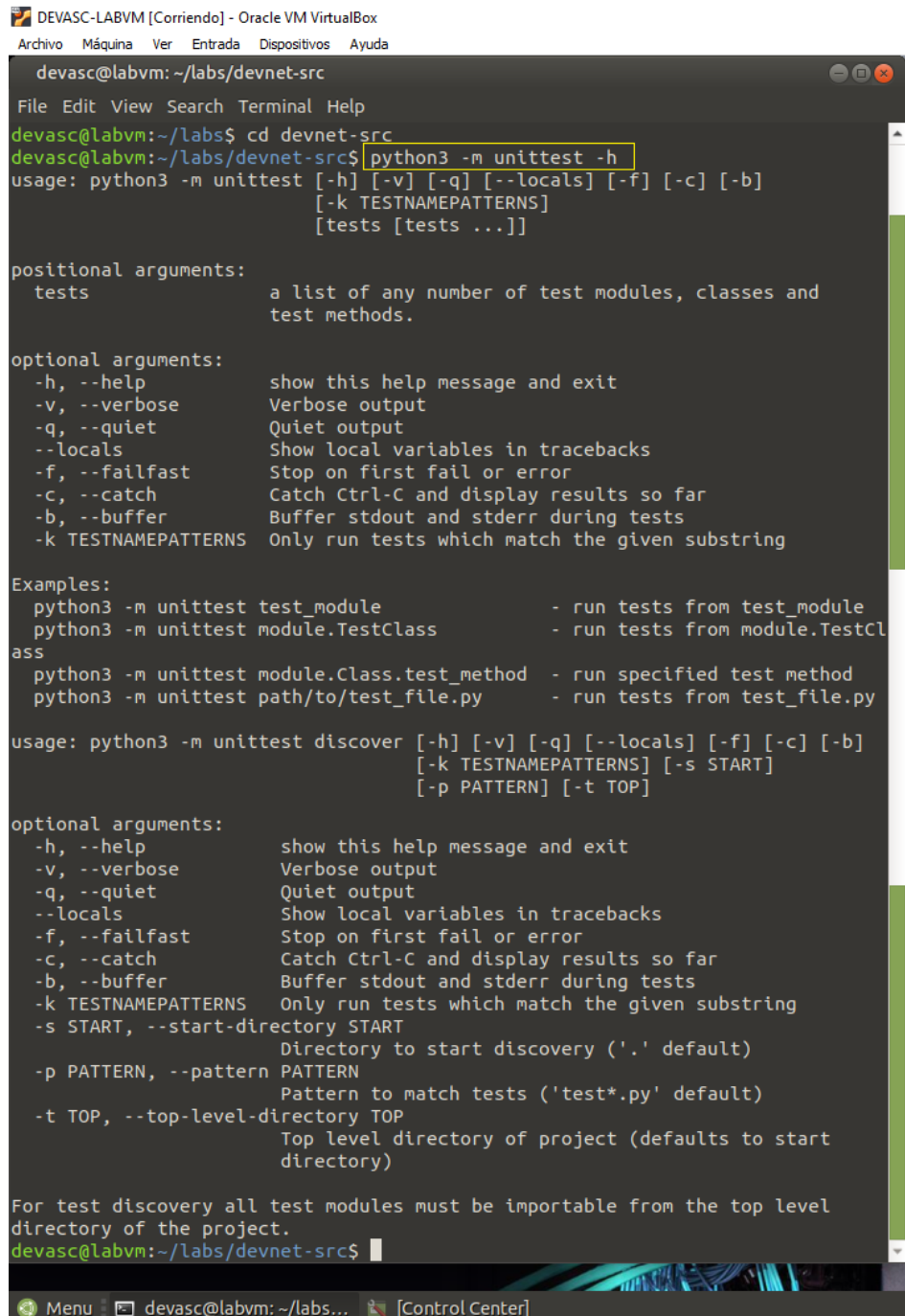
interfaz gráfica, una interfaz textual o devolver un valor especial para indicar los resultados de la ejecución de las pruebas.

Para ejecutar pruebas con más detalle pasando el indicador **-v**:

➤ `python3 -m unittest -v test_module`

Para obtener una lista de todas las opciones de la línea de comandos:

➤ `python3 -m unittest -h`



```
DEVASC-LABVM [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda

devasc@labvm: ~/labs/devnet-src
File Edit View Search Terminal Help
devasc@labvm:~/labs$ cd devnet-src
devasc@labvm:~/labs/devnet-src$ python3 -m unittest -h
usage: python3 -m unittest [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                           [-k TESTNAMEPATTERNS]
                           [tests [tests ...]]

positional arguments:
  tests                a list of any number of test modules, classes and
                        test methods.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose          Verbose output
  -q, --quiet           Quiet output
  --locals              Show local variables in tracebacks
  -f, --failfast         Stop on first fail or error
  -c, --catch           Catch Ctrl-C and display results so far
  -b, --buffer          Buffer stdout and stderr during tests
  -k TESTNAMEPATTERNS  Only run tests which match the given substring

Examples:
  python3 -m unittest test_module           - run tests from test_module
  python3 -m unittest module.TestClass      - run tests from module.TestCL
ass
  python3 -m unittest module.Class.test_method - run specified test method
  python3 -m unittest path/to/test_file.py  - run tests from test_file.py

usage: python3 -m unittest discover [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                                    [-k TESTNAMEPATTERNS] [-s START]
                                    [-p PATTERN] [-t TOP]

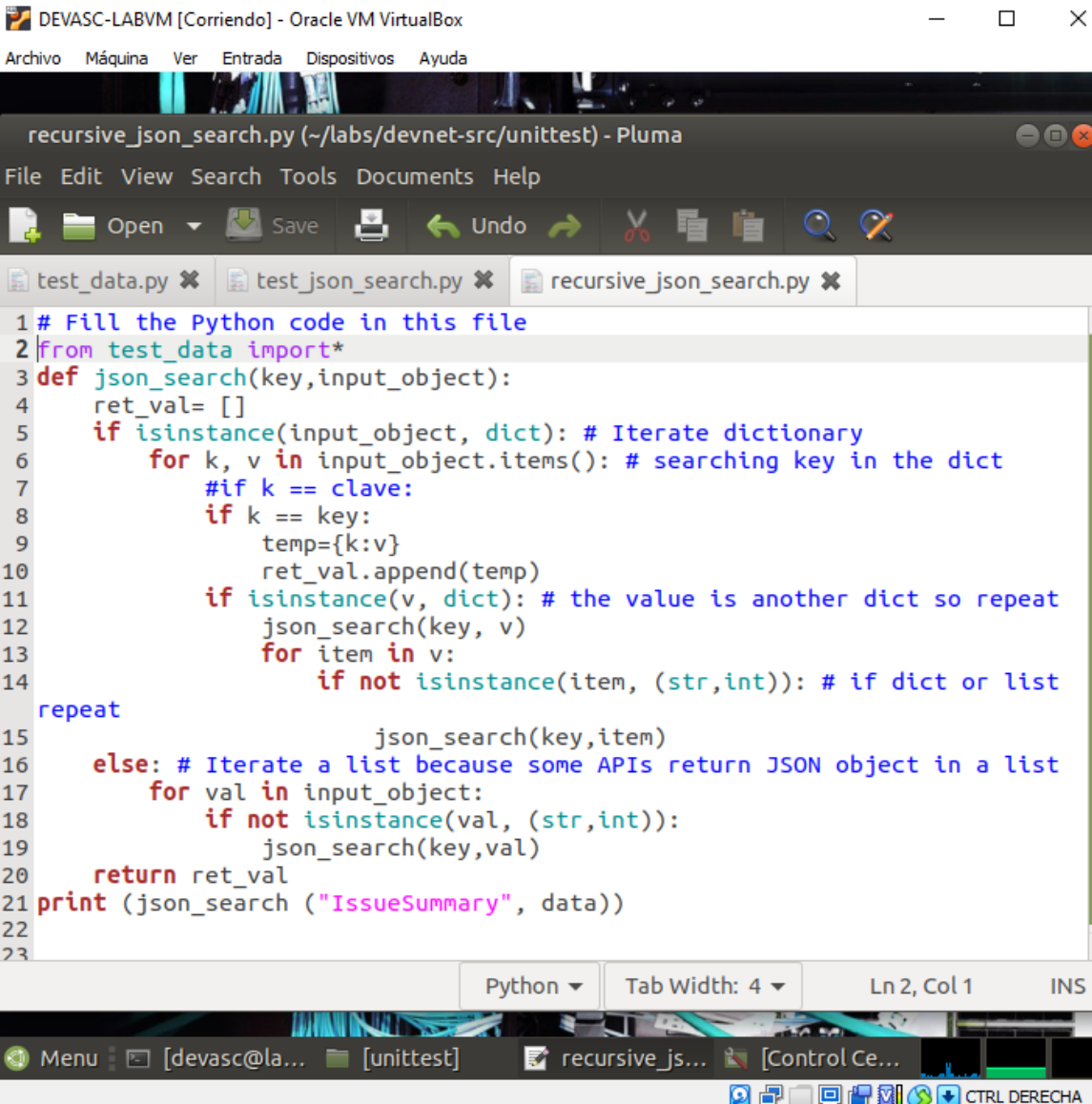
optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose          Verbose output
  -q, --quiet           Quiet output
  --locals              Show local variables in tracebacks
  -f, --failfast         Stop on first fail or error
  -c, --catch           Catch Ctrl-C and display results so far
  -b, --buffer          Buffer stdout and stderr during tests
  -k TESTNAMEPATTERNS  Only run tests which match the given substring
  -s START, --start-directory START
                        Directory to start discovery ('.' default)
  -p PATTERN, --pattern PATTERN
                        Pattern to match tests ('test*.py' default)
  -t TOP, --top-level-directory TOP
                        Top level directory of project (defaults to start
                        directory)

For test discovery all test modules must be importable from the top level
directory of the project.
devasc@labvm:~/labs/devnet-src$
```

PARTE 3: Probar una función de Python con unittest.

En esta parte, usará unittest para probar una función que realiza una búsqueda recursiva de un objeto JSON. La función devuelve valores etiquetados con una clave dada. Los programadores a menudo necesitan realizar este tipo de operación en objetos JSON devueltos por llamadas API.

Creando la función `json_search()` en el archivo `recursive_json_search.py` nuestra función recibe como parámetro una “clave” y un “objeto” JSON y devolverá una lista de pares clave/valor.



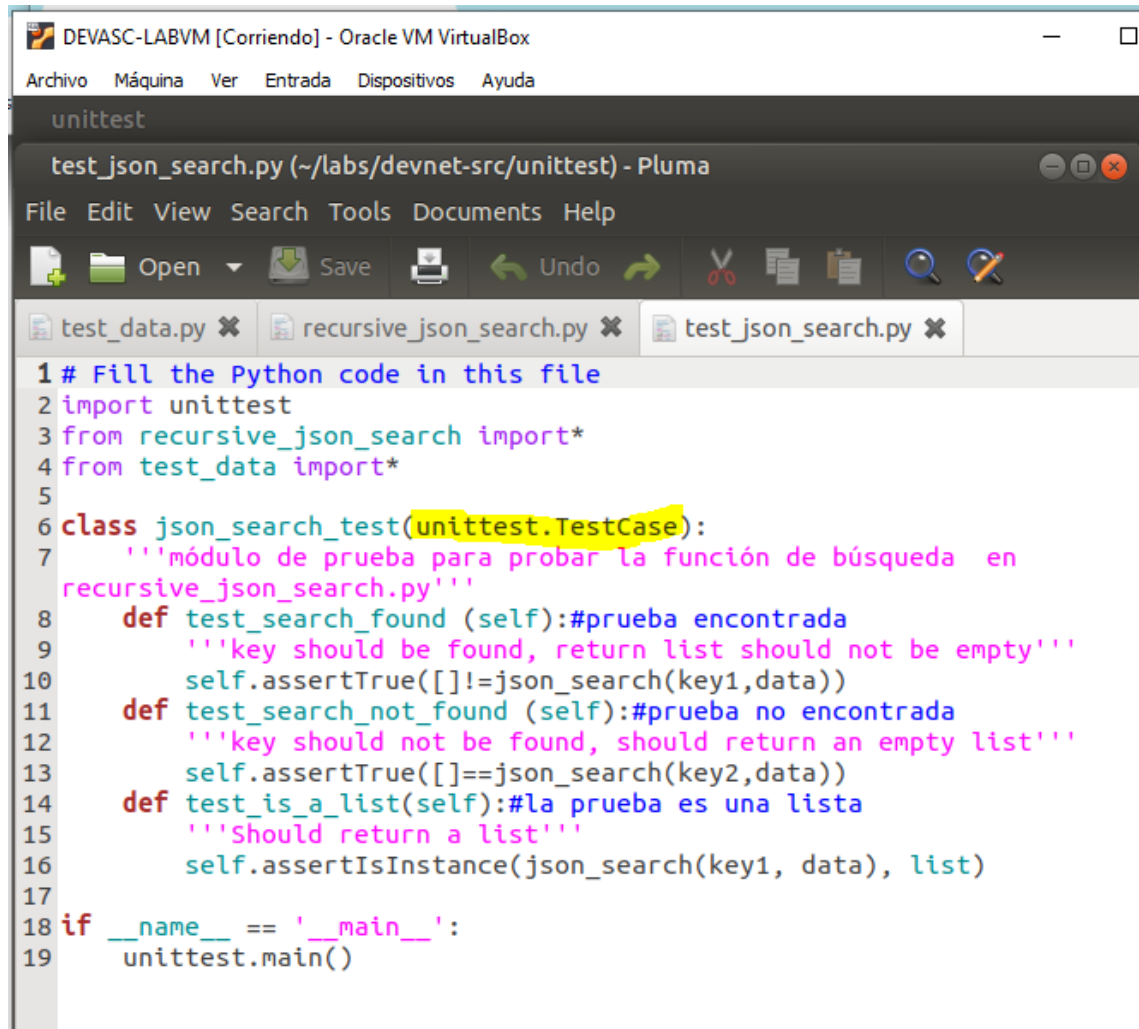
The screenshot shows a virtual machine window titled "DEVASC-LABVM [Corriendo] - Oracle VM VirtualBox". Inside, a terminal window displays the execution of a Python script. The script, `recursive_json_search.py`, is shown in a code editor window titled "recursive_json_search.py (~/.labs/devnet-src/unittest) - Pluma". The code defines a recursive function `json_search` that searches for a key in a JSON object and returns a list of values. The script is executed using `python3 recursive_json_search.py`, and the output is an empty list `[]`.

```
1 # Fill the Python code in this file
2 from test_data import*
3 def json_search(key,input_object):
4     ret_val= []
5     if isinstance(input_object, dict): # Iterate dictionary
6         for k, v in input_object.items(): # searching key in the dict
7             #if k == clave:
8             if k == key:
9                 temp={k:v}
10                ret_val.append(temp)
11            if isinstance(v, dict): # the value is another dict so repeat
12                json_search(key, v)
13                for item in v:
14                    if not isinstance(item, (str,int)): # if dict or list
15                        json_search(key,item)
16            else: # Iterate a list because some APIs return JSON object in a list
17                for val in input_object:
18                    if not isinstance(val, (str,int)):
19                        json_search(key,val)
20        return ret_val
21 print (json_search ("IssueSummary", data))
22
23
```

```
devasc@labvm:~/labs/devnet-src/unittest$ ls
__pycache__ recursive_json_search.py test_data.py test_json_search.py
devasc@labvm:~/labs/devnet-src/unittest$ python3 recursive_json_search.py
[]
devasc@labvm:~/labs/devnet-src/unittest$
```

Abrimos el archivo **test_json_search.py** para crear pruebas unitarias que probaran si la función trabaja según lo previsto.

Creamos una clase **json_search_test**, ahí creamos la subclase **TestCase** del **unittest** framework.



```
1 # Fill the Python code in this file
2 import unittest
3 from recursive_json_search import*
4 from test_data import*
5
6 class json_search_test(unittest.TestCase):
7     '''módulo de prueba para probar la función de búsqueda en
8     recursive_json_search.py'''
9     def test_search_found (self):#prueba encontrada
10         '''key should be found, return list should not be empty'''
11         self.assertTrue([]!=json_search(key1,data))
12     def test_search_not_found (self):#prueba no encontrada
13         '''key should not be found, should return an empty list'''
14         self.assertTrue([]==json_search(key2,data))
15     def test_is_a_list(self):#la prueba es una lista
16         '''Should return a list'''
17         self.assertIsInstance(json_search(key1, data), list)
18
19 if __name__ == '__main__':
20     unittest.main()
```

Ejecutamos la prueba para ver los resultados del **TestCase**..

Resultado	Descripción
ok (.)	La prueba pasa.
Fail (F)	La prueba no pasa y genera una excepción AssertionError .
Error (E)	La prueba genera cualquier excepción que no es AssertionError

```
test data.py x test json_search.py x recursive json_search.py x
1 # devasc@labvm: ~/labs/devnet-src/unittest
2 File Edit View Search Terminal Help
3 f devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
4 []
5 c.F.
6
7 =====
8 FAIL: test_search_found (__main__.json_search_test)
9 key should be found, return list should not be empty
10 -----
11 Traceback (most recent call last):
12   File "test_json_search.py", line 10, in test_search_found
13     self.assertTrue([]!=json_search(key1,data))
14   AssertionError: False is not true
15 -----
16 Ran 3 tests in 0.001s
17
18 FAILED (failures=1)
19 devasc@labvm:~/labs/devnet-src/unittest$
```

Para ejecutar pruebas con más detalle pasamos el indicador **-v** :

➤ `python3 -m unittest -v test_json_search.py`

```
test data.py x test json_search.py x recursive json_search.py x
1 # devasc@labvm: ~/labs/devnet-src/unittest
2 File Edit View Search Terminal Help
3 f devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest -v test_json_
4 search
5 []
6 c
7 test_is_a_list (test_json_search.json_search_test)
8 Should return a list ... ok
9 test_search_found (test_json_search.json_search_test)
10 key should be found, return list should not be empty ... FAIL
11 test_search_not_found (test_json_search.json_search_test)
12 key should not be found, should return an empty list ... ok
13
14 =====
15 FAIL: test_search_found (test_json_search.json_search_test)
16 key should be found, return list should not be empty
17 -----
18 Traceback (most recent call last):
19   File "/home/devasc/labs/devnet-src/unittest/test_json_search.py", line 1
20 0, in test_search_found
21     self.assertTrue([]!=json_search(key1,data))
22   AssertionError: False is not true
23 -----
24 Ran 3 tests in 0.001s
```


Corregimos el **primer** error en el script **recursive_json_search.py**
Ahora debemos encontrar la clave que le pasamos por parámetro a la función **json_search()** para que no me retorne una lista vacía. Eso lo hacemos inicializando la lista **ret_val=[]** fuera de la función **json_search()**, así evitamos que cada vez que llame recursivamente nos devuelva la lista vacía.
En este caso es CaseSensitive, porque en el pdf del laboratorio esta "IssueSummary" y no lo encontraba y me devolvió lista vacía.

```
recursive_json_search.py (~/.devnet-src/unittest) - Pluma
devasc@labvm: ~/.devnet-src/unittest
File Edit View Search Terminal Help
devasc@labvm:~/.devnet-src/unittest$ python3 recursive_json_search.py
[]
devasc@labvm:~/.devnet-src/unittest$ python3 recursive_json_search.py
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
devasc@labvm:~/.devnet-src/unittest$
```

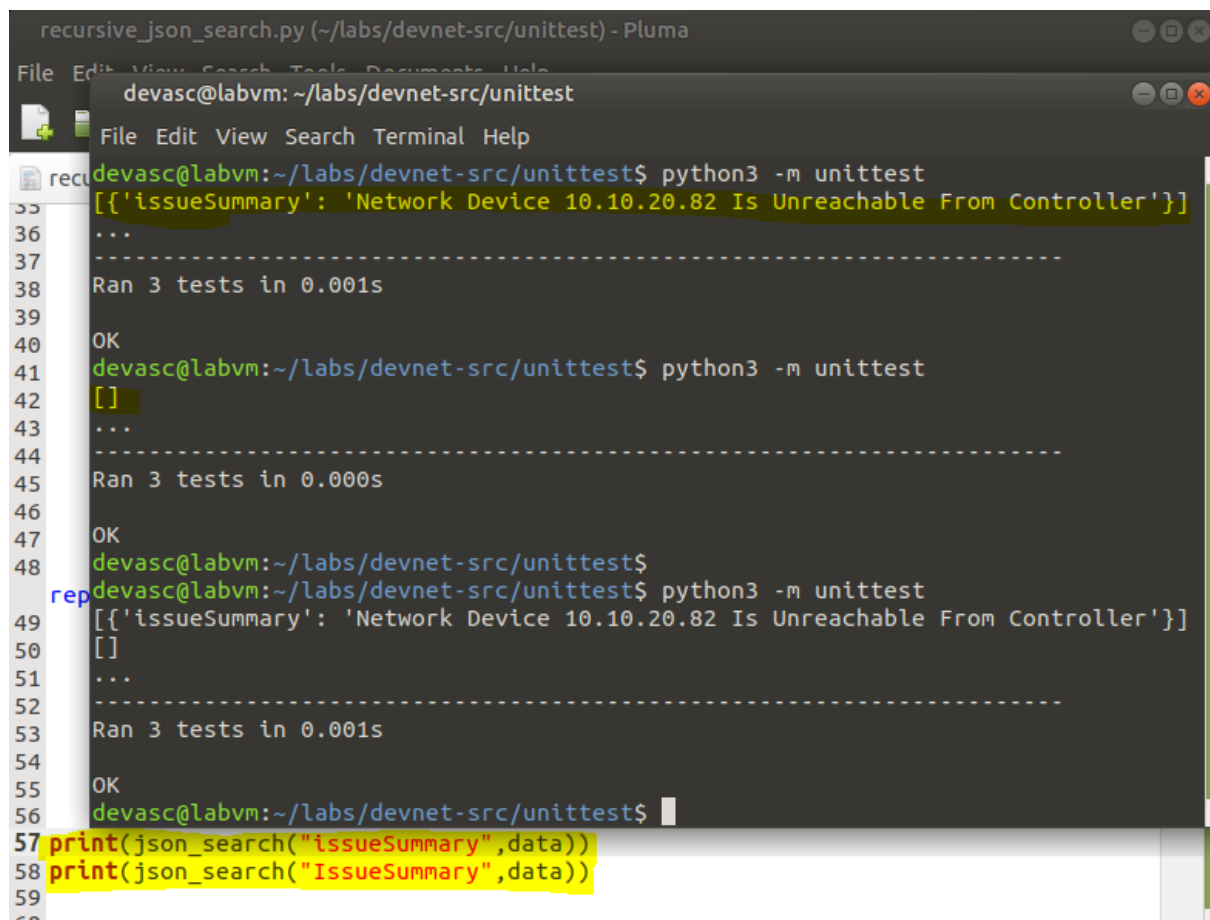
Ahora ejecutamos el unittest, **test_json_search.py** pero notamos que falla la tercera prueba.

```
devasc@labvm:~/.devnet-src/unittest$ python3 recursive_json_search.py
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
devasc@labvm:~/.devnet-src/unittest$
devasc@labvm:~/.devnet-src/unittest$ python3 -m unittest test_json_search.py
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
..F
=====
FAIL: test_search_not_found (test_json_search.json_search_test)
key should not be found, should return an empty list
-----
Traceback (most recent call last):
  File "/home/devasc/labs/devnet-src/unittest/test_json_search.py", line 13, in test_search_not_found
    self.assertTrue([]==json_search(key2,data))
AssertionError: False is not true
-----
Ran 3 tests in 0.001s

FAILED (failures=1)
devasc@labvm:~/.devnet-src/unittest$
```

Corregimos el **segundo** error en el script **recursive_json_Search.py** una vez corregido la variable global **ret_val=[]**, ahora Refactorizaremos la función **json_search()** para mejorar la estructura interna sin alterar la funcionalidad.

En este caso nuestra función distingue entre Mayuscula y minuscula, porque si buscamos de la siguiente forma "IssueSummary" no lo encontrará pero pasará la prueba.



```
recursive_json_search.py (~/.devnet-src/unittest) - Pluma
File Edit View Search Tools Documents Help
devasc@labvm: ~/labs/devnet-src/unittest
File Edit View Search Terminal Help
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
...
-----
37
38 Ran 3 tests in 0.001s
39
40 OK
41 devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
42 []
43 ...
44 -----
45 Ran 3 tests in 0.000s
46
47 OK
48 devasc@labvm:~/labs/devnet-src/unittest$
49 rep devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
50 [{}{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
51 []
52 ...
53 -----
54 Ran 3 tests in 0.001s
55
56 OK
57 devasc@labvm:~/labs/devnet-src/unittest$
58 print(json_search("issueSummary",data))
59 print(json_search("IssueSummary",data))
60
```


Bibliografía:

[1] *unittest — Unit testing framework — Python 3.11.3 documentation.* (n.d.).

Python Docs. Retrieved April 16, 2023, from

<https://docs.python.org/3/library/unittest.html>