

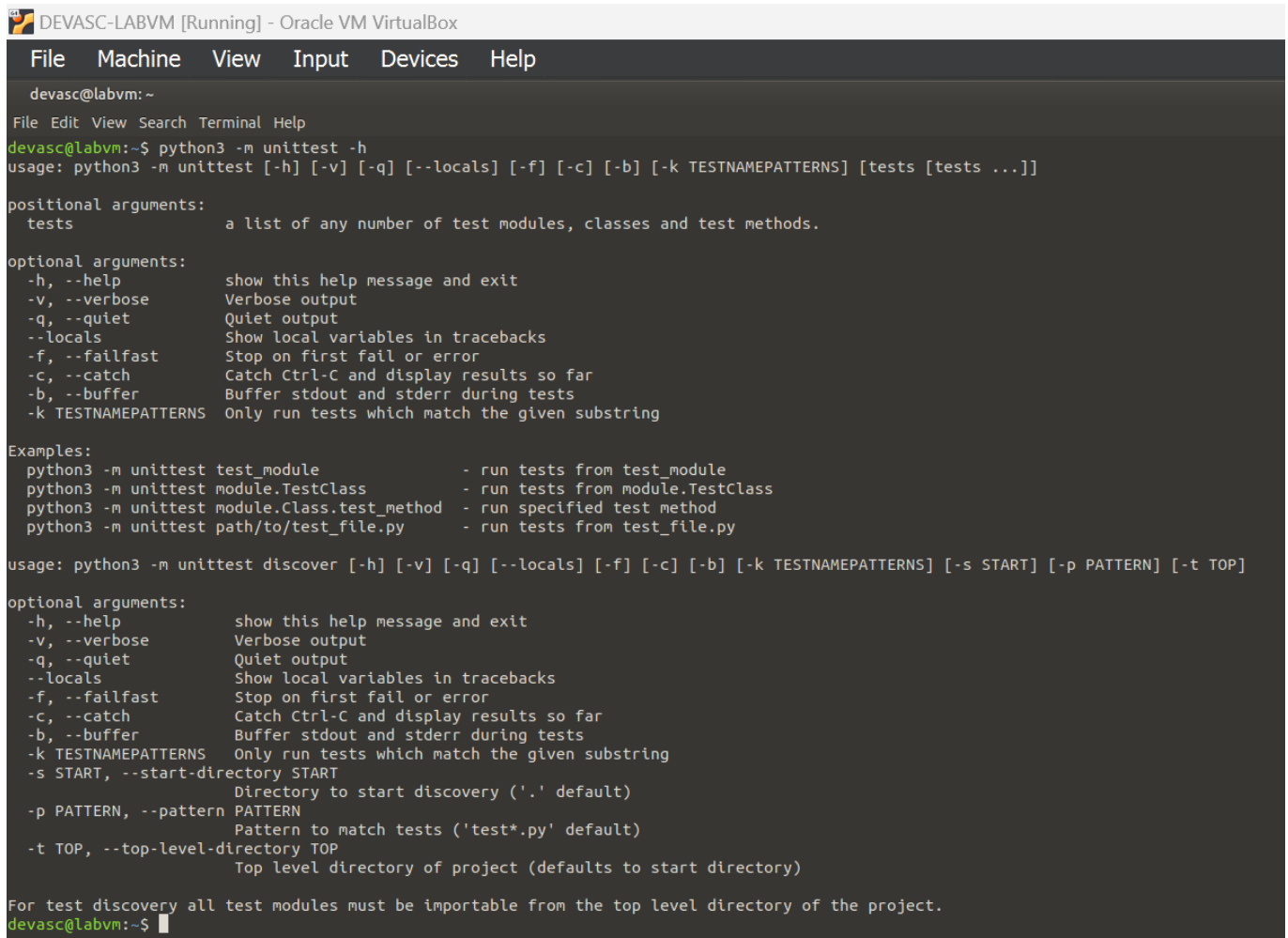
# Laboratorio 3a - Crear una prueba unitaria de Python

## Instrucciones

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

Hecho!

Parte 2: Explorar las opciones en el framework unittest



```
DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm:~
File Edit View Search Terminal Help
devasc@labvm:~$ 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.TestClass
  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:~$
```

Parte 3: Probar una función Python con unittest

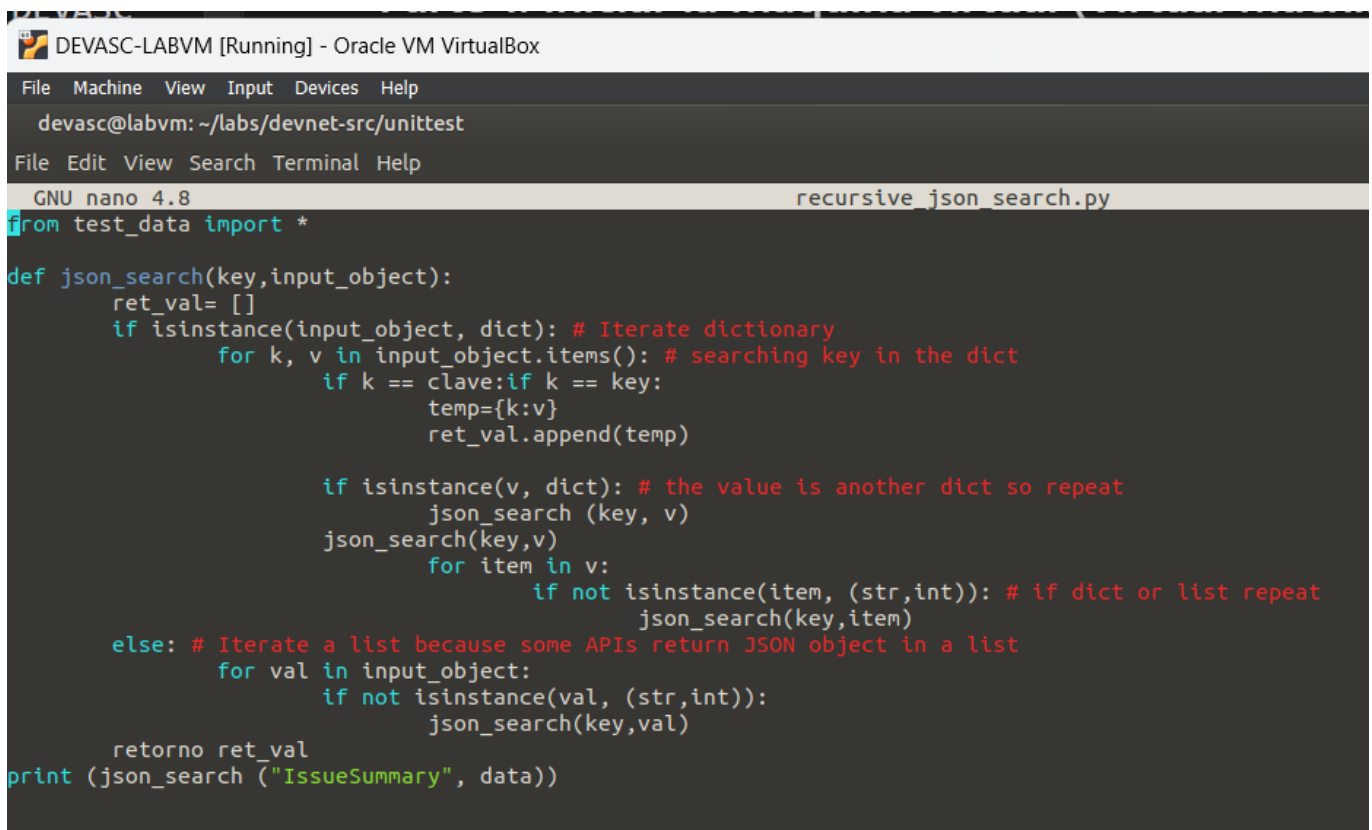
### Paso 1: Revisar el archivo test\_data.py

```
devasc@labvm:~$ cd ~/labs/devnet-src/unittest/
devasc@labvm:~/labs/devnet-src/unittest$ cat test_data.py
key1 = "issueSummary"
key2 = "XY&^$#@!1234%^&"

data = {
    "id": "AWcvsjx864kVeDHDi2gB",
    "instanceId": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197",
    "category": "Warn",
    "status": "NEW",
    "title": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197"
```

## Paso 2: Crear la función json\_search() que se va a probar

```
nano recursive_json_search.py
```



```
DEVASC-LABVM [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/unittest
File Edit View Search Terminal Help
GNU nano 4.8 recursive_json_search.py
from test_data import *

def json_search(key,input_object):
    ret_val= []
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == clave:if k == key:
                temp={k:v}
                ret_val.append(temp)

            if isinstance(v, dict): # the value is another dict so repeat
                json_search (key, v)
            json_search(key,v)
            for item in v:
                if not isinstance(item, (str,int)): # if dict or list repeat
                    json_search(key,item)
    else: # Iterate a list because some APIs return JSON object in a list
        for val in input_object:
            if not isinstance(val, (str,int)):
                json_search(key,val)

    retorno ret_val
print (json_search ("IssueSummary", data))
```

Como es de esperar no funciona

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 recursive_json_search.py
File "recursive_json_search.py", line 7
    if k == clave:if k == key:
                  ^
SyntaxError: invalid syntax
```

## Paso 3: Crear algunas pruebas unitarias

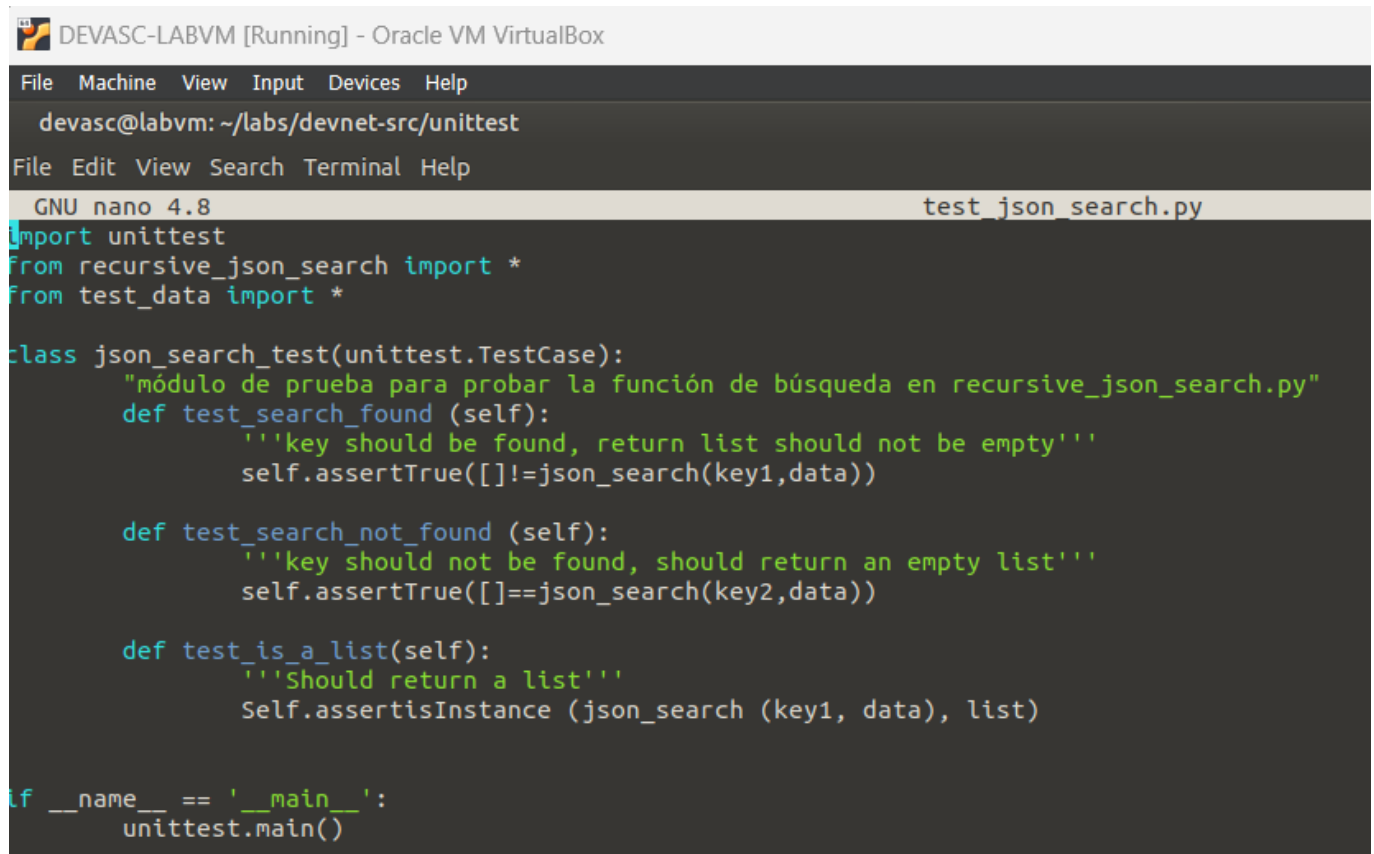
Se arreglo los errores de sintáxis de recursive\_json\_search.py

```
from test_data import *

def json_search(key,input_object):
    ret_val= []
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == key:
                temp={k:v}
                ret_val.append(temp)

            if isinstance(v, dict): # the value is another dict so repeat
                json_search (key, v)
            for item in v:
                if not isinstance(item, (str,int)): # if dict or list repeat
                    json_search(key,item)
    else: # Iterate a list because some APIs return JSON object in a list
        for val in input_object:
            if not isinstance(val, (str,int)):
                json_search(key,val)
    return ret_val
print (json_search ("IssueSummary", data))
```

Prueba unitaria:



The screenshot shows a terminal window titled "DEVASC-LABVM [Running] - Oracle VM VirtualBox". The terminal displays the output of running unit tests for the recursive\_json\_search.py module. The tests are defined in test\_json\_search.py and include test\_search\_found, test\_search\_not\_found, and test\_is\_a\_list. The output shows that all tests passed successfully.

```
File Machine View Input Devices Help
devasc@labvm: ~/labs/devnet-src/unittest
File Edit View Search Terminal Help
GNU nano 4.8 test_json_search.py
import unittest
from recursive_json_search import *
from test_data import *

class json_search_test(unittest.TestCase):
    "módulo de prueba para probar la función de búsqueda en recursive_json_search.py"
    def test_search_found (self):
        '''key should be found, return list should not be empty'''
        self.assertTrue([]!=json_search(key1,data))

    def test_search_not_found (self):
        '''key should not be found, should return an empty list'''
        self.assertTrue([]==json_search(key2,data))

    def test_is_a_list(self):
        '''Should return a list'''
        Self.assertisInstance (json_search (key1, data), list)

if __name__ == '__main__':
    unittest.main()
```

**Paso 4: Ejecutar la prueba para ver los resultados iniciales**

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
[]
EF.
=====
ERROR: test_is_a_list (__main__.json_search_test)
Should return a list
-----
Traceback (most recent call last):
  File "test_json_search.py", line 17, in test_is_a_list
    Self.assertIsInstance (json_search (key1, data), list)
NameError: name 'Self' is not defined
=====
FAIL: test_search_found (__main__.json_search_test)
key should be found, return list should not be empty
-----
Traceback (most recent call last):
  File "test_json_search.py", line 9, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true
-----
Ran 3 tests in 0.001s

FAILED (failures=1, errors=1)
```

Como vemos hay errores.

#### Paso 5: Investigar y corregir el primer error

#### Paso 6: Ejecutar la prueba de nuevo

```
EF.
=====
ERROR: test_is_a_list (__main__.json_search_test)
Should return a list
-----
Traceback (most recent call last):
  File "test_json_search.py", line 17, in test_is_a_list
    Self.assertIsInstance (json_search (key1, data), list)
NameError: name 'Self' is not defined
=====
FAIL: test_search_found (__main__.json_search_test)
key should be found, return list should not be empty
-----
Traceback (most recent call last):
  File "test_json_search.py", line 9, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true
-----
Ran 3 tests in 0.001s
```

#### Paso 7: Investigar y corregir el segundo error

```

devasc@labvm: ~/labs/devnet-src/unittest
File Edit View Search Terminal Help
GNU nano 4.8 recursive_json_search.py
from test_data import *

def json_search(clave, input_object):
    """
    Buscar una clave del objeto JSON, no obtener nada si la clave no se encuentra
    key: "keyword" a buscar, distingue entre mayúsculas y minúsculas
    input_object: objeto JSON a analizar, test_data.py en este caso
    inner_function() está haciendo la búsqueda recursiva
    devolver una lista de par clave: valor
    """
    ret_val = []

    def inner_function(clave, input_object):
        if isinstance(input_object, dict): # Iterate dictionary
            for k, v in input_object.items(): # searching key in the dict
                if k == clave:
                    temp = {k: v}
                    ret_val.append(temp)
                if isinstance(v, dict): # the value is another dict so repeat
                    inner_function(clave, v)
                elif isinstance(v, list):
                    for item in v:
                        if not isinstance(item, (str, int)): # if dict or list repeat
                            inner_function(clave, item)
            else: # Itere una lista porque algunas API devuelven un objeto JSON en una lista
                for val in input_object:
                    if not isinstance(val, (str, int)):
                        inner_function(clave, val)

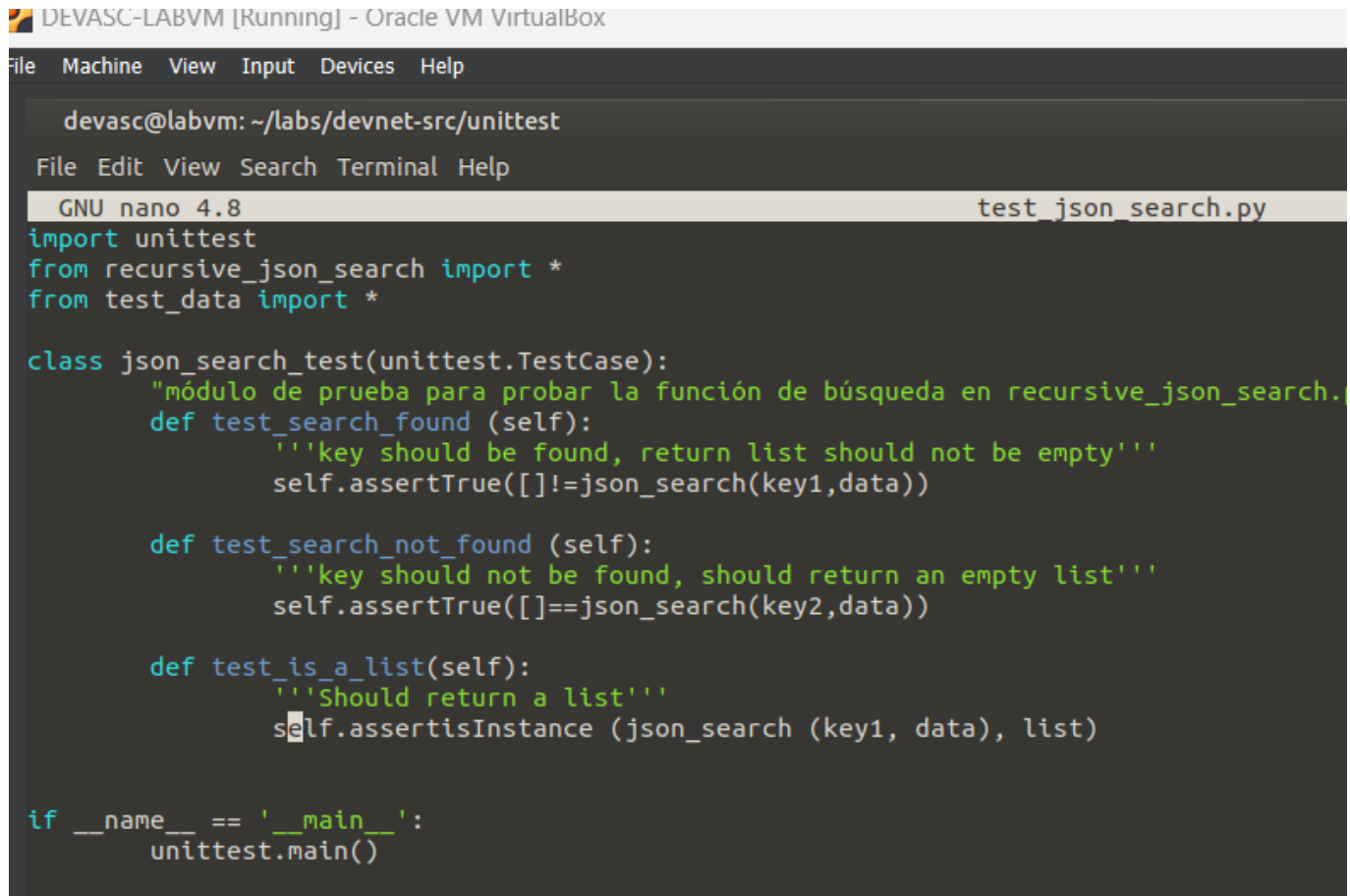
    inner_function(clave, input_object)
    return ret_val

print(json_search("IssueSummary", data))

```

## Extra pasos

Me di cuenta que el error que me salía era porque puse una mayúscula **S** en lugar de **s** en `test_json_search.py`, en `self.asertisInstance`. Además que es `self.asertIsInstance`.



```
devasc@labvm: ~/labs/devnet-src/unittest
File Edit View Search Terminal Help

GNU nano 4.8 test_json_search.py
import unittest
from recursive_json_search import *
from test_data import *

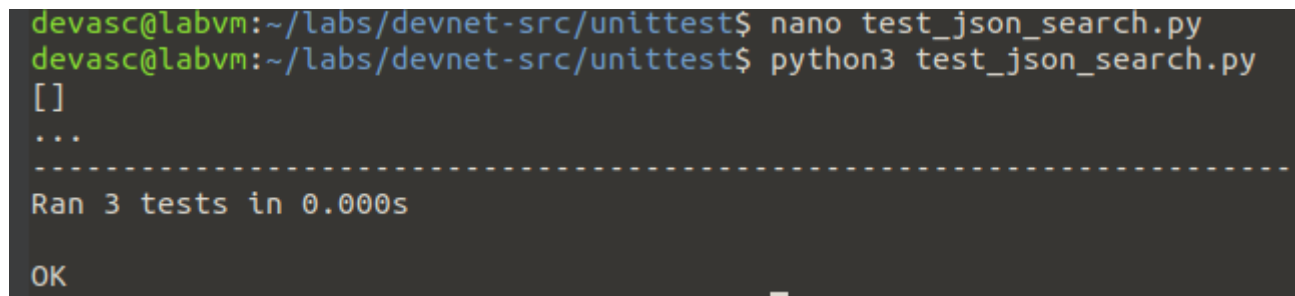
class json_search_test(unittest.TestCase):
    "módulo de prueba para probar la función de búsqueda en recursive_json_search."
    def test_search_found (self):
        '''key should be found, return list should not be empty'''
        self.assertTrue([]!=json_search(key1,data))

    def test_search_not_found (self):
        '''key should not be found, should return an empty list'''
        self.assertTrue([]==json_search(key2,data))

    def test_is_a_list(self):
        '''Should return a list'''
        self.assertIsInstance (json_search (key1, data), list)

if __name__ == '__main__':
    unittest.main()
```

Luego de esos 2 cambios, la prueba unitaria funcionó.



```
devasc@labvm:~/labs/devnet-src/unittest$ nano test_json_search.py
devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
[]
...
-----
Ran 3 tests in 0.000s

OK
```

## Reflexión

En este laboratorio se aprendió a crear pruebas unitarias en Python utilizando el framework unittest. Se exploraron los conceptos de clases de prueba, métodos de prueba, y manejo de aserciones. Además, se trabajó con estructuras de datos JSON y se aplicaron técnicas de recursividad para buscar información dentro de los datos.

Este conocimiento es esencial para garantizar la calidad del software, ya que permite verificar que el código funcione correctamente bajo diferentes escenarios y condiciones. Las pruebas unitarias son fundamentales para asegurar que los cambios en el código no introduzcan nuevos errores y para garantizar que el software cumpla con los requisitos esperados.