Лабораторна робота № 2. Написання unit-тестів на мові Python

Метою даної лабораторної роботи є отримання базових навичок по написанню unit-тестів на мові Python.

Завдання на лабораторну роботу

- Запустити DEVASC VM
- Перевірити інсталяцію Python
- Перевірити роботу PIP і налаштувати віртуальні середовища Python
- Перевірити роботу python-фреймворка для тестування unittest

Хід роботи:

1. Запустити DEVASC VM.

2. Перевірити встановлену версію Python

- Перевірте версію Python, яка вже встановлена у DEVASC VM , за допомогою команди *python3 -V*.

```
devasc@labvm:~$ python3 -V
Python 3.8.2
devasc@labvm:~$
```

- Щоб побачити каталог для локального середовища Python, скористайтеся командою *which python3*.

```
devasc@labvm:~$ which python3
/usr/bin/python3
devasc@labvm:~$
```

3. Робота з PIP і Python Virtual Environments

PIP (Pip Installs Packages). Для уникнення ситуацій, коли окремі пакети, їх версії і залежності будуть конкурувати для різних проектів хорошою практикою вважається використання Python Virtual Environments. Це дає можливість встановити лише ті пакети, які необхідні для конкретного проекту у цьому віртуальному середовищі. Таким чином, контролюються перелік і версія пакетів встановлених в віртуальному оточенні з певними параметрами. Щоб встановити віртуальне середовище Python, скористайтеся інструментом venv у Python 3, а потім активуйте віртуальне середовище, як показано в наступних кроках.

Створення Python virtual environment:

- В DEVASC VM, перейдіть в каталог *labs/devnet-src/python*

```
devasc@labvm:~$ cd labs/devnet-src/python/
devasc@labvm:~/labs/devnet-src/python$
```

- Введіть наступну команду, щоб використовувати інструмент venv для створення Python virtual environment з іменем *devfun*. Ключ *-m* наказує Python запустити модуль venv.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m venv devfun
devasc@labvm:~/labs/devnet-src/python$
```

Активація та перевірка Python virtual environment.

- Активуйте віртуальне середовище. Підказка змінюється, щоб вказати назву середовища, в якому ви зараз працюєте (в даному випадку *devfun*). Використавши команду *pip3 install*, система встановлюватиме лише пакети для активного віртуального середовища.

```
devasc@labvm:~/labs/devnet-src/python$ source devfun/bin/activate
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

- Виконайте команду *pip3 freeze*, щоб переконатися, що в середовищі devfun наразі не містить додаткових пакетів Python.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

- Тепер встановіть пакет requests у середовищі *devfun*.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 install requests
Collecting requests
  Downloading requests-2.23.0-py2.py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 290 kB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
  Downloading urllib3-1.25.9-py2.py3-none-any.whl (126 kB)
     |████████████████████████████████| 126 kB 1.7 MB/s
Collecting idna<3,>=2.5
  Downloading idna-2.9-py2.py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 18.3 MB/s
Collecting certifi>=2017.4.17
  Downloading certifi-2020.4.5.1-py2.py3-none-any.whl (157 kB)
     |████████████████████████████████| 157 kB 19.8 MB/s
Collecting chardet<4,>=3.0.2
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
     |████████████████████████████████| 133 kB 59.2 MB/s
Installing collected packages: urllib3, idna, certifi, chardet, requests
Successfully installed certifi-2020.4.5.1 chardet-3.0.4 idna-2.9 requests-2.23.0
urllib3-1.25.9
  (devfun) devasc@labvm:~/labs/devnet-src/python$
```

- Повторно введіть команду *pip3 freeze*, щоб побачити пакети, які зараз встановлені в середовищі *devfun*. Примітка. Ваш список пакетів і номери версій можуть дещо відрізнятися.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
certifi==2020.4.5.1
chardet==3.0.4
idna==2.10
requests==2.24.0
urllib3==1.25.9
(devfun) devasc@labvm:~/labs/devnet-src/python$
```

- Щоб деактивувати віртуальне середовище та повернутися до системи, введіть команду *deactivate*.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$
```

Перевірка пакетів встановлених в системному оточенні

- Введіть загальносистемну команду *python3 -m pip freeze*, щоб побачити, які пакети встановлені в системному середовищі. Примітка. Оскільки Python 3 викликається за допомогою наступної команди, ви використовуєте команду *pip* замість *pip3*.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m pip freeze
ansible==2.9.4
apache-libcloud==2.8.0
appdirs==1.4.3
argcomplete==1.8.1
astroid==2.3.3
bcrypt==3.1.7
blinker==1.4
certifi==2019.11.28
<output omitted>
xmltodict==0.12.0
zipp==1.0.0
devasc@labvm:~/labs/devnet-src/python$
```

- Якщо ви хочете швидко знайти версію встановленого пакета, передайте вихід команді *grep*. Введіть наступну команду, щоб побачити поточну версію пакета запитів

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m pip freeze | grep requests
requests==2.22.0
requests-kerberos==0.12.0
requests-ntlm==1.1.0
requests-toolbelt==0.9.1
requests-unixsocket==0.2.0
devasc@labvm:~/labs/devnet-src/python$
```

Перенесення віртуального оточення.

Вивід команди *pip3 freeze* має певний формат. Ви можете використовувати всі перераховані залежності, щоб інші люди, які хочуть працювати над тим самим проектом, що й ви, могли отримати те саме середовище, що й ваше. Розробник може створити файл, наприклад, *requirements.txt*, використовуючи команду *pip3 freeze > requirements.txt*. Тоді інший розробник може з іншого активованого віртуального середовища використовувати команду *pip3 install -r requirements.txt* для встановлення пакетів, необхідних для проекту.

- Знову активуйте *devfun virtual environment*.

```
devasc@labvm:~/labs/devnet-src/python$ source devfun/bin/activate
(devfun) devasc@labvm:
```

- Запишіть результат виведення команди *pip3 freeze* в файл *requirements.txt*.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze >
requirements.txt
```

- Вимкніть віртуальне середовище *devfun*. Ви можете скористатися командою *ls*, щоб побачити, що файл *requirements.txt* знаходиться в каталозі */python*.

```
(devfun) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$ ls
devfun        file-access-input.py  if-acl.py       requirements.txt
devices.txt   file-access.py        if-vlan.py      while-loop.py
devnew        hello-world.py        person-info.py
```

- Створіть і активуйте нове virtual environment з назвою *devnew*.

```
devasc@labvm:~/labs/devnet-src/python$ python3 -m venv devnew
devasc@labvm:~/labs/devnet-src/python$ source devnew/bin/activate
(devnew) devasc@labvm:~/labs/devnet-src/python$
```

- Використовуйте команду *pip3 install -r requirements.txt*, щоб встановити ті самі пакети, які встановлені у віртуальному середовищі *devfun*.

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ pip3 install -r
requirements.txt
Collecting certifi==2020.4.5.1
  Using cached certifi-2020.4.5.1-py2.py3-none-any.whl (157 kB)
Collecting chardet==3.0.4
  Using cached chardet-3.0.4-py2.py3-none-any.whl (133 kB)
Collecting idna==2.9
  Using cached idna-2.9-py2.py3-none-any.whl (58 kB)
Requirement already satisfied: pkg-resources==0.0.0 in ./devnew/lib/python3.8/site-
packages (from -r requirements.txt (line 4)) (0.0.0)
Collecting requests==2.23.0
  Using cached requests-2.23.0-py2.py3-none-any.whl (58 kB)
Collecting urllib3==1.25.9
  Using cached urllib3-1.25.9-py2.py3-none-any.whl (126 kB)
Installing collected packages: certifi, chardet, idna, urllib3, requests
Successfully installed certifi-2020.4.5.1 chardet-3.0.4 idna-2.9 requests-2.23.0
urllib3-1.25.9
(devnew) devasc@labvm:~/labs/devnet-src/python$
```

- Під час введення *pip3 freeze* у середовищі *devnew* ви повинні побачити наступний результат.

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ pip3 freeze
certifi==2020.4.5.1
chardet==3.0.4
idna==2.9
pkg-resources==0.0.0
requests==2.23.0
urllib3==1.25.9
(devnew) devasc@labvm:~/labs/devnet-src/python$
```

- Деактивуйте *devnew* virtual environment

```
(devnew) devasc@labvm:~/labs/devnet-src/python$ deactivate
devasc@labvm:~/labs/devnet-src/python$
```

4. Проведення тестування за допомогою фреймворку unittest

В цьому завданні буде використаний фреймворк для юніт-тестування unittest для перевірки функції, яка виконує рекурсивний пошук об’єкта JSON. Функція повертає значення, позначені заданим ключем. Розробникам часто доводиться виконувати подібні операції над об’єктами JSON, які повертаються викликами API. У цьому тесті будуть використовуватися три файли, зведені в наступній таблиці:

Файл	Опис
recursive_json_search.py	Файл, що містить функцію json_search(), що тестуватиметься
test_data.py	Дані, в яких відбуватиметься пошук за допомогою json_search()
test_json_search.py	Файл, що буде створений для тестування функції json_search() в файлі recursive_json_search.py

- Перегляньте файл *test_data.py file*

  Відкрийте файл *~/labs/devnet-src/unittest/test_data.py* і перегляньте його вміст. Ці дані JSON є типовими для даних, які повертаються викликом API DNA Center Cisco. Зразок даних досить складний, щоб бути хорошим тестом. Наприклад, він має суміш даних типів dict і list.

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

data = {
  "id": "AWcvsjx864kVeDHDi2gB",
  "instanceId": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197",
  "category": "Warn",
  "status": "NEW",
  "timestamp": 1542693469197,
  "severity": "P1",
  "domain": "Availability",
  "source": "DNAC",
  "priority": "P1",
  "type": "Network",
  "title": "Device unreachable",
  "description": "This network device leaf2.abc.inc is unreachable from controll
er. The device role is ACCESS.",
  "actualServiceId": "10.10.20.82",
  "assignedTo": "",
  "enrichmentInfo": {
    "issueDetails": {
      "issue": [
        {
--More--(12%)
```

- Напишіть функцію *json_search()*, що буде тестуватися.

Наша функція повинна очікувати ключ і об’єкт JSON як вхідні параметри і повертати список відповідних пар ключ/значення. Ось поточна версія функції, яку потрібно перевірити, щоб перевірити, чи працює вона належним чином. Мета цієї функції — спочатку імпортувати дані тесту. Потім вона шукає дані, які відповідають ключовим змінним у файлі *test_data.py*. Якщо вона знайде відповідність, функція додасть відповідні дані до списку. Функція *print()* в кінці друкує вміст списку для першої змінної *key1 = "issueSummary"*.

```
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)
            elif isinstance(v, list): # it's a list
                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))
```

- Відкрийте файл *~/labs/devnet-src/unittest/recursive_json_search.py*
- Скопіюйте код вище у файл і збережіть його.
- Запустіть код. Ви не повинні отримати жодних помилок і вивести *[ ]*, що вказує на порожній список. Якби функція *json_search()* була написана правильно (а це не так), це означало б, що немає даних із ключем *«issueSummary»*, повідомленим даними JSON, що повертаються API Cisco DNA Center. Іншими словами, немає проблем, про які слід повідомляти.

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

- Але як дізнатися, що функція *json_search()* працює належним чином? Ви можете відкрити файл test_data.py і знайти ключ *«issueSummary»*, як показано нижче. Якби ви це зробили, ви справді виявили б, що є проблема. Це невеликий набір даних і відносно простий рекурсивний пошук. Проте виробничі дані та код рідко бувають такими простими. Тому тестування коду є життєво важливим для швидкого пошуку та виправлення помилки у вашому коді.

```
<output omitted>
      "issue": [
        {
          "issueId": "AWcvsjx864kVeDHDi2gB",
          "issueSource": "Cisco DNA",
          "issueCategory": "Availability",
          "issueName": "snmp_device_down",
          "issueDescription": "This network device leaf2.abc.inc is
unreachable from controller. The device role is ACCESS.",
          "issueEntity": "network_device",
          "issueEntityValue": "10.10.20.82",
          "issueSeverity": "HIGH",
          "issuePriority": "",
          "issueSummary": "Network Device 10.10.20.82 Is Unreachable From
Controller",
          "issueTimestamp": 1542693469197,
          "suggestedActions": [
            {
<output omitted>
```

5. Написання модульних тестів для перевірки коректності роботи функції.

- Відкрийте файл *~ labs/devnet-src/unittest/test_json_search.py*
- Скопіюйте в нього наступний код

```
import unittest

from recursive_json_search import *
from test_data import *

class json_search_test(unittest.TestCase):
    '''test module to test search function in
`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()
```

6. Запуск тесту для перевірки результатів

- Запустіть тестовий сценарій у його поточному стані, щоб побачити, які результати він зараз повертає.

По-перше, ви бачите порожній список. По-друге, ви бачите .F. підсвічено у виводі. Крапка (.) означає, що тест пройдено, а F означає, що тест не пройшов. Отже, перший тест пройшов, другий — провалено, а третій — пройшов.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 test_json_search.py
 []
.F.
======================================================================
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 11, in test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

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

- Щоб перерахувати кожен тест і його результати, запустіть сценарій ще раз у unittest з опцією *-v*. Зверніть увагу, що вам не потрібне розширення *.py* для сценарію *test_json_search.py*. Ви можете побачити, що ваш метод тестування *test_search_found* не працює. Примітка: Python не обов'язково запускає ваші тести по порядку. Тести виконуються в алфавітному порядку на основі назв методів тестування.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest -v
test_json_search
[]
test_is_a_list (test_json_search.json_search_test)
Should return a list ... ok
test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty ... FAIL
test_search_not_found (test_json_search.json_search_test)
key should not be found, should return an empty list ... ok

======================================================================
FAIL: test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/devasc/labs/devent-src/unittest/test_json_search.py", line 11, in
test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

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

7. Дослідження і виправлення помилки в файлі *recursive_json_search.py*

FAIL, вказує, що ключ не знайдено. Чому? Якщо ми подивимося на текст рекурсивної функції, ми побачимо, що оператор *ret_val=[ ]* багаторазово виконується при кожному виклику функції. Це призводить до того, що функція очищає список і втрачає накопичені результати з оператора *ret_val.append(temp)*, який додається до списку, створеного *ret_val=[ ]*.

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

- В файлі *recursive_json_search.py* веремістіть *ret_val=[ ]* з функції, щоб ітерація щоразу не перезаписувала накопичений список.

```
ret_val=[]
def json_search(key,input_object):
```

- Збережіть і запустіть скрипт. Ви повинні отримати наступний вивід, який підтверджує, що ви вирішили проблему. Тобто список більше не порожній після запуску сценарію.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 recursive_json_search.py
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
devasc@labvm:~/labs/devnet-src/unittest$
```

8. Запуск файлу і перевірка виправлення помилок.

- Запустіть unittest знову без параметра *-v*, щоб перевірити, чи повертає *test_json_search* будь-які помилки. Як правило, ви не використовуватимете параметр *-v*, щоб мінімізувати вивід в консолі та зробити тести швидшими. На початку звіту ви можете побачити *..F*, що означає, що третій тест не пройшов. Також зверніть увагу, що список все ще друкується. Ви можете припинити цю поведінку, видаливши функцію *print()* у файлі *resursive_json_search.py*.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest test_json_search
[{'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 14, 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:~/labs/devnet-src/unittest$
```

- Відкрийте файл *test_data.py* і знайдіть *issueSummary*, яке є значенням для *key1*. Ви повинні знайти його двічі, але лише один раз в об'єкті даних JSON. Але якщо ви шукаєте значення для *key2*, тобто *XY&^$#*@!1234%^&*, ви знайдете його лише вгорі, де воно визначено, оскільки його немає в об'єкті JSON. Третій тест - це перевірка, щоб переконатися, що його там немає. У третьому тестовому коментарі вказано, що ключ повинен не знайдено, має повернути порожній список. Однак функція повернула непорожній список.

9. Пошук і виправлення другої помилки в recursive_json_search.py

Перегляньте код *recursive_json_search.py* ще раз. Зверніть увагу, що *ret_val* тепер є глобальною змінною після того, як ви виправили її на попередньому кроці. Це означає, що його значення зберігається при кількох викликах функції

json_search(). Це хороший приклад того, чому використання глобальних змінних у функціях є поганою практикою.

Щоб вирішити цю проблему, оберніть функцію *json_search()* зовнішньою функцією. Видаліть наявну функцію *json_search()* і замініть її на модифіковану функцію нижче:

```
from test_data import *
def json_search(key,input_object):
    """
    Search a key from JSON object, get nothing back if key is not found
    key : "keyword" to be searched, case sensitive
    input object : JSON object to be parsed, test data.py in this case
    inner function() is actually doing the recursive search
    return a list of key:value pair
    """
    ret_val=[]
    def inner_function(key,input_object):
        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
                    inner_function(key,v)
                elif isinstance(v, list):
                    for item in v:
                        if not isinstance(item, (str,int)): # if dict or list
repeat
                            inner_function(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)):
                    inner_function(key,val)
    inner_function(key,input_object)
    return ret_val
print(json_search("issueSummary",data))
```

- Збережіть файл і запустіть *unittest*. Ім'я якогось конкретного файлу не потрібно тому, що функція unittest Test Discovery запускає будь-який знайдений локальний файл, ім’я якого починається з test. Ви повинні отримати наступний результат. Зверніть увагу, що тепер усі тести пройшли, і список для ключа *«issueSummary»* заповнено. Ви можете безпечно видалити функцію *print()*, оскільки вона зазвичай не використовується, коли цей тест об’єднується з іншими тестами для більшого тесту.

```
devasc@labvm:~/labs/devnet-src/unittest$ python3 -m unittest
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From Controller'}]
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
devasc@labvm:~/labs/devnet-src/unittest$
```

Запишіть висновки по виконаній роботі.