Advanced DevOps

- Sanjiva Bennur

Linux-Python – Part2

Python Data Structure

```
# Python list
ages = [05, 19, 57]
print(ages)
#list containing strings, numbers and another list
student = ['Test', 22, 'NewDelhi', [4, 5]]
print(student)
# Python set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)
# Tuple integer type
student_id = (112, 114, 116, 118, 115)
print('Student ID:', student_id)
# creating a dictionary
country_capitals = {
   "Germany": "Berlin",
   "Canada": "Ottawa",
   "England": "London"
}
# printing the dictionary
print(country_capitals)
```

Creating Python Virtual Environment

Please follow below procedure to create python virtual environment

update packages

sudo apt update

to install python virtual environment

sudo apt-get install python3-venv

create a folder - test

go to test folder

install virtual environment for project folder

sudo python3 -m venv venv

#activate virtual environment by typing below command on terminal

source venv/bin/activate

The directory shows small icon of directory name at left side

Python Error Exception - Runtime

```
# Example of an exception
n = 10
try:
    res = n / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError:
    print("Can't be divided by zero!")
================================================================================
try:
    n = 0
    res = 100 / n
except ZeroDivisionError:
    print("You can't divide by zero!")
except ValueError:
    print("Enter a valid number!")
else:
    print("Result is", res)
finally:
    print("Execution complete.")

================================================================================
```

Error Exception – Customised

```
def set(age):
    if age < 0:
        raise ValueError("Age cannot be negative.")
    print(f"Age set to {age}")
try:
    set(-5)
except ValueError as e:
    print€
```

While Program

```
num =0
while num< 50:
        num = num + 1 # Incrementing 1,2,3,4,5
        print("num =", num)
```

Create Python Package – Part-1

```
Steps to Create Python Package
1: create new directory
mkdir mypackage
2: Add Modules
# module1.py
def add(a, b):
return a + b
# module2.py
def subtract(a, b):
return a - b
3: Create the __init__.py File
# __init__.py
from .module1 import add
from .module2 import subtract
4: Create the setup.py File
from setuptools import setup, find_packages
setup(
name="mypackage",
version="0.1",
packages=find_packages(),
install_requires=[],
)
```

Create Python Package – Part-2

```
5: Build the Package
to build the package, run the following command in the directory containing setup.py:
python setup.py sdist bdist_wheel
This above command will create a dist directory containing the distribution files.
6: Upload to PyPI
To upload your package to PyPI, you need to create an account on PyPI. Once you have an account,
install twine:
pip install twine
Then, upload your package using twine:
twine upload dist/*
You will be prompted to enter your PyPI username and password.
7: Install Your Package
pip install mypackage
Create program
import mypackage
print(mypackage.add(25, 25)) # Output: 50
print(mypackage.subtract(50, 20)) # Output: 30
```

Python task schedule

```
import schedule
import time
def print_message():
    print("Task executed at:", time.strftime("%H:%M:%S"))
# Schedule task to run every 5 seconds
schedule.every(5).seconds.do(print_message)
# Keep the program running to allow scheduled tasks to execute
while True:
    schedule.run_pending()
    time.sleep(1)
```

Python Task Schedule

```
def print_message():
    print("Task executed at:", time.strftime("%H:%M:%S"))
# Schedule task to run every 5 seconds
schedule.every(5).seconds.do(print_message)
# Keep the program running to allow scheduled tasks to execute
while True:
    schedule.run_pending()
    time.sleep(1)
```

Python Task Automation

```
import shutil
import os
source_folder = '/path/to/source'
destination_folder = '/path/to/destination'
# Copy all files from source to destination
shutil.copytree(source_folder, destination_folder)
```

Health Status of Website

```
Health checks using Python script
import requests
def check_website_health(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            print(f"Website {url} is up and running!")
        else:
            print(f"Website {url} returned status code {response.status_code}.")
    except requests.exceptions.RequestException as e:
        print(f"Error checking {url}: {e}")
# Example usage
check_website_health(" Any website ")
```

Python Health Check – Alert – Disk space available

```
import shutil
def check_disk_space(threshold):
    total, used, free = shutil.disk_usage("/")
    free_percent = (free / total) * 100
    if free_percent < threshold:
        print(f"Warning: Disk space is below {threshold}%! Only {free_percent:.2f}% free.")
    else:
        print(f"Disk space is sufficient: {free_percent:.2f}% free.")
# Example usage
check_disk_space(20)  # Alert if free space is below 20%
```

Available CPU Usage – Health Check

```
import psutil
def check_cpu_usage(threshold):
    cpu_usage = psutil.cpu_percent(interval=1)
    if cpu_usage > threshold:
        print(f"Warning: CPU usage is above {threshold}%! Current usage: {cpu_usage}%.")
    else:
        print(f"CPU usage is normal: {cpu_usage}%.")
# Example usage
check_cpu_usage(80)  # Alert if CPU usage is above 80%
```

File and directory commands

```
import os
# change directory
os.chdir('cd directoryname')
print(os.getcwd())
==========================================
import os
# list all sub-directories
print(os.listdir())
==========================================
os.mkdir('test')
os.listdir()
['test']
==========================================
import os
os.listdir()
['test']
# rename a directory
os.rename('test','new_one')
os.listdir()
['new_one']
```

API Creation & Testing using Curl Command

```
Install Flask

pip install Flask
from flask import Flask, jsonify, request
app = Flask(__name__)
@app.route('/')
def home():
    return "Welcome to the API!"
@app.route('/api/data', methods=['GET'])
def get_data():
    data = {
        'name': 'test',
        'age': 30,
        'city': 'Bengaluru'
    }
    return jsonify(data)
@app.route('/api/data', methods=['POST'])
def add_data():
    new_data = request.get_json()
    return jsonify(new_data), 201
if __name__ == '__main__':
    app.run(debug=True)
```

API Testing Using CURL Command

```
Open your browser and go to http://127.0.0.1:5000/ to see the welcome message.
curl http://127.0.0.1:5000/api/data
curl -X POST -H "Content-Type: application/json" -d '{"name": "test1", "age": 35, "city":
"Mysore"}' http://127.0.0.1:5000/api/data
```

Python security Best Practices

- Use Virtual Environments
- Always use virtual environments to isolate dependencies for your project. This prevents conflicts between packages and ensures you're not accidentally using outdated or vulnerable libraries from other projects.
- Keep Dependencies Up-to-Date
- Regularly update third-party libraries using `pip` and avoid using outdated versions that may have known vulnerabilities.

Python security Best Practices

- Validate and Sanitize User Input
- Always validate and sanitize inputs from users to prevent SQL injection, XSS (Cross-Site Scripting), and other forms of injection attacks.
- Avoid Hardcoding Sensitive Information
- Never hardcode sensitive data such as passwords, API keys, or tokens directly in the codebase. Use environment variables or secret management tools like `dotenv` for local development or cloud services for production.

Python security Best Practices

- Use Secure Password Hashing
- Always hash passwords before storing them. Use modern cryptographic algorithms like **bcrypt**, **argon2**, or **PBKDF2** instead of SHA or MD5.
- Limit Access and Permissions
- Apply the principle of least privilege by limiting the permissions of users, processes, and services to only what is strictly necessary.
- Secure Communication (Use HTTPS)

Python security Best Practices

- Ensure that sensitive data is transmitted securely over HTTPS rather than HTTP. When using Python libraries for web requests (e.g., `requests`),
- Logging and Monitoring
- Properly log events but avoid logging sensitive information such as passwords or personal data. Use libraries like `logging` in Python for structured logging.
- Implement monitoring for abnormal behavior or potential security breaches.

Python security Best Practices

- Use Security Headers
- For web applications, use security headers to protect against attacks such as XSS, clickjacking, and CSRF. Headers like `Content-Security-Policy`, `Strict-Transport-Security`, and `X-Content-Type-Options` can add an extra layer of defense.
- **Enforce Multi-Factor Authentication (MFA)**
- For systems that require user login, ensure that multi-factor authentication (MFA) is enforced, adding an extra layer of security.

Python – Networking – Server.py

```
# first of all import the socket library
import socket
# next create a socket object
s = socket.socket()
print ("Socket successfully created")
# reserve a port on your computer in our
port = 40674
# Next bind to the port
# coming from other computers on the network
s.bind(('', port))
print ("socket binded to %s" %(port))
# put the socket into listening mode
s.listen(5)
print ("socket is listening")
# a forever loop until we interrupt it or
# an error occurs
while True:
    # Establish connection with client.
    c, addr = s.accept()
    print ('Got connection from', addr )
    # send a thank you message to the client.
    c.send(b'Thank you for connecting')
    # Close the connection with the client
    c.close()
```

Start Server

- # start the server
- python server.py

- # keep the above terminal open
- # now open another terminal and type:

- telnet localhost 12345

Client Server – client.py

```
# Import socket module
import socket
# Create a socket object
s = socket.socket()
# Define the port on which you want to connect
port = 40674
# connect to the server on local computer
s.connect(('127.0.0.1', port))
# receive data from the server
print(s.recv(1024))
# close the connection
s.close()
```

Assignment: Python Advanced Shell Scripts

1) Write Python script for user account management
2) Write Python script for disk usage notification
3) Write Python script for automatically update system packages
4) Write Python script for System Load Resource usage monitoring
5) Write Python script to Monitor SSL Certification Expiry dates
6) Write python script for setting up file system permissions
7) Write python script to monitor key device components
8) Write Python script to manage System access controls
9) Write Python Script to analyse System logs
10) Write Python Script to monitor user account daily work logged hours

Thank you

Question and Answer

EXCELSOFT