# Python commands for DevOps

# 1. File Operations

- **Read a file:**

  python

  ```
  with open('file.txt', 'r') as file:
      content = file.read()
      print(content)
  ```

- **Write to a file:**

  python

  ```
  with open('output.txt', 'w') as file:
      file.write('Hello, DevOps!')
  ```

# 2. Environment Variables

- **Get an environment variable:**

  python

  ```
  import os

  db_user = os.getenv('DB_USER')
  print(db_user)
  ```

- **Set an environment variable:**

  python

  ```
  import os

  os.environ['NEW_VAR'] = 'value'
  ```

# 3. Subprocess Management

- **Run shell commands:**

python

import subprocess

result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
print(result.stdout)

# 4. API Requests

- **Make a GET request:**

  python

  import requests

  response = requests.get('https://api.example.com/data')
  print(response.json())

# 5. JSON Handling

- **Read JSON from a file:**

  python

  import json

  with open('data.json', 'r') as file:
      data = json.load(file)
      print(data)

- **Write JSON to a file:**

  python

  import json

  data = {'name': 'DevOps', 'type': 'Workflow'}
  with open('output.json', 'w') as file:
      json.dump(data, file, indent=4)

# 6. Logging

- **Basic logging setup:**

  python

  import logging

  logging.basicConfig(level=logging.INFO)
  logging.info('This is an informational message')

# 7. Working with Databases

- **Connect to a SQLite database:**

  python

  import sqlite3

  conn = sqlite3.connect('example.db')
  cursor = conn.cursor()
  cursor.execute('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)')
  conn.commit()
  conn.close()

# 8. Automation with Libraries

- **Using Paramiko for SSH connections:**

  python

  import paramiko

  ssh = paramiko.SSHClient()
  ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
  ssh.connect('hostname', username='user', password='password')

  stdin, stdout, stderr = ssh.exec_command('ls')
  print(stdout.read().decode())
  ssh.close()

# 9. Error Handling

- **Try-except block:**

python

```
try:
    # code that may raise an exception
    risky_code()
except Exception as e:
    print(f'Error occurred: {e}')
```

# 10. Docker Integration

- **Using the docker package to interact with Docker:**

  python

  ```
  import docker

  client = docker.from_env()
  containers = client.containers.list()
  for container in containers:
      print(container.name)
  ```

# 11. Working with YAML Files

- **Read a YAML file:**

  python

  ```
  import yaml

  with open('config.yaml', 'r') as file:
      config = yaml.safe_load(file)
      print(config)
  ```

- **Write to a YAML file:**

  python

  ```
  import yaml

  data = {'name': 'DevOps', 'version': '1.0'}
  with open('output.yaml', 'w') as file:
      yaml.dump(data, file)
  ```

# 12. Parsing Command-Line Arguments

- **Using argparse:**

  python

  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  parser.add_argument('--num', type=int, help='an integer for the accumulator')

  args = parser.parse_args()
  print(args.num)

# 13. Monitoring System Resources

- **Using psutil to monitor system resources:**

  python

  import psutil

  print(f"CPU Usage: {psutil.cpu_percent()}%")
  print(f"Memory Usage: {psutil.virtual_memory().percent}%")

# 14. Handling HTTP Requests with Flask

- **Basic Flask API:**

  python

  from flask import Flask, jsonify

  app = Flask(__name__)

  @app.route('/health', methods=['GET'])
  def health_check():
      return jsonify({'status': 'healthy'})

  if __name__ == '__main__':
      app.run(host='0.0.0.0', port=5000)

# 15. Creating Docker Containers

- **Using the Docker SDK to create a container:**

  python

  ```
  import docker

  client = docker.from_env()
  container = client.containers.run('ubuntu', 'echo Hello World', detach=True)
  print(container.logs())
  ```

# 16. Scheduling Tasks

- **Using schedule for task scheduling:**

  python

  ```
  import schedule
  import time

  def job():
      print("Running scheduled job...")

  schedule.every(1).minutes.do(job)

  while True:
      schedule.run_pending()
      time.sleep(1)
  ```

# 17. Version Control with Git

- **Using GitPython to interact with Git repositories:**

  python

  ```
  import git

  repo = git.Repo('/path/to/repo')
  repo.git.add('file.txt')
  repo.index.commit('Added file.txt')
  ```

# 18. Email Notifications

- **Sending emails using smtplib:**

  python

  ```
  import smtplib
  from email.mime.text import MIMEText

  msg = MIMEText('This is the body of the email')
  msg['Subject'] = 'Email Subject'
  msg['From'] = 'you@example.com'
  msg['To'] = 'recipient@example.com'

  with smtplib.SMTP('smtp.example.com', 587) as server:
      server.starttls()
      server.login('your_username', 'your_password')
      server.send_message(msg)
  ```

# 19. Creating Virtual Environments

- **Creating and activating a virtual environment:**

  python

  ```
  import os
  import subprocess

  # Create virtual environment
  subprocess.run(['python3', '-m', 'venv', 'myenv'])

  # Activate virtual environment (Windows)
  os.system('myenv\\Scripts\\activate')

  # Activate virtual environment (Linux/Mac)
  os.system('source myenv/bin/activate')
  ```

# 20. Integrating with CI/CD Tools

- **Using the requests library to trigger a Jenkins job:**

  python

```
import requests

url = 'http://your-jenkins-url/job/your-job-name/build'
response = requests.post(url, auth=('user', 'token'))
print(response.status_code)
```

# 21. Database Migration

- **Using Alembic for database migrations:**

  bash

  ```
  alembic revision -m "initial migration"
  alembic upgrade head
  ```

# 22. Testing Code

- **Using unittest for unit testing:**

  python

  ```
  import unittest

  def add(a, b):
      return a + b

  class TestMathFunctions(unittest.TestCase):
      def test_add(self):
          self.assertEqual(add(2, 3), 5)

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

# 23. Data Transformation with Pandas

- **Using pandas for data manipulation:**

  python

  ```
  import pandas as pd

  df = pd.read_csv('data.csv')
  ```

```
df['new_column'] = df['existing_column'] * 2
df.to_csv('output.csv', index=False)
```

# 24. Using Python for Infrastructure as Code

- **Using boto3 for AWS operations:**

  python

  ```
  import boto3

  ec2 = boto3.resource('ec2')
  instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])
  for instance in instances:
      print(instance.id, instance.state)
  ```

# 25. Web Scraping

- **Using BeautifulSoup to scrape web pages:**

  python

  ```
  import requests
  from bs4 import BeautifulSoup

  response = requests.get('http://example.com')
  soup = BeautifulSoup(response.content, 'html.parser')
  print(soup.title.string)
  ```

# 26. Using Fabric for Remote Execution

- **Running commands on a remote server:**

  python

  ```
  from fabric import Connection
  ```

```
conn = Connection(host='user@hostname', connect_kwargs={'password':
'your_password'})
conn.run('uname -s')
```

# 27. Automating AWS S3 Operations

- **Upload and download files using boto3:**

  python

  ```
  import boto3

  s3 = boto3.client('s3')

  # Upload a file
  s3.upload_file('local_file.txt', 'bucket_name', 's3_file.txt')

  # Download a file
  s3.download_file('bucket_name', 's3_file.txt', 'local_file.txt')
  ```

# 28. Monitoring Application Logs

- **Tail logs using tail -f equivalent in Python:**

  python

  ```
  import time

  def tail_f(file):
      file.seek(0, 2)  # Move to the end of the file
      while True:
          line = file.readline()
          if not line:
              time.sleep(0.1)  # Sleep briefly
              continue
          print(line)

  with open('app.log', 'r') as log_file:
      tail_f(log_file)
  ```

# 29. Container Health Checks

- **Check the health of a running Docker container:**

  python

  ```
  import docker

  client = docker.from_env()
  container = client.containers.get('container_id')
  print(container.attrs['State']['Health']['Status'])
  ```

# 30. Using requests for Rate-Limited APIs

- **Handle rate limiting in API requests:**

  python

  ```
  import requests
  import time

  url = 'https://api.example.com/data'
  while True:
      response = requests.get(url)
      if response.status_code == 200:
          print(response.json())
          break
      elif response.status_code == 429:  # Too Many Requests
          time.sleep(60)  # Wait a minute before retrying
      else:
          print('Error:', response.status_code)
          break
  ```

# 31. Docker Compose Integration

- **Using docker-compose in Python:**

  python

  ```
  import os
  import subprocess

  # Start services defined in docker-compose.yml
  subprocess.run(['docker-compose', 'up', '-d'])

  # Stop services
  subprocess.run(['docker-compose', 'down'])
  ```

# 32. Terraform Execution

- **Executing Terraform commands with subprocess:**

  python

  ```
  import subprocess

  # Initialize Terraform
  subprocess.run(['terraform', 'init'])

  # Apply configuration
  subprocess.run(['terraform', 'apply', '-auto-approve'])
  ```

# 33. Working with Prometheus Metrics

- **Scraping and parsing Prometheus metrics:**

  python

  ```
  import requests

  response = requests.get('http://localhost:9090/metrics')
  metrics = response.text.splitlines()
  ```

```
for metric in metrics:
    print(metric)
```

# 34. Using pytest for Testing

- **Simple test case with pytest:**

  python

  ```
  def add(a, b):
      return a + b

  def test_add():
      assert add(2, 3) == 5
  ```

# 35. Creating Webhooks

- **Using Flask to create a simple webhook:**

  python

  ```
  from flask import Flask, request

  app = Flask(__name__)

  @app.route('/webhook', methods=['POST'])
  def webhook():
      data = request.json
      print('Received data:', data)
      return 'OK', 200

  if __name__ == '__main__':
      app.run(port=5000)
  ```

# 36. Using Jinja2 for Configuration Templates

- **Render configuration files with Jinja2:**

python

from jinja2 import Template

template = Template('Hello, {{ name }}!')
rendered = template.render(name='DevOps')
print(rendered)

# 37. Encrypting and Decrypting Data

- **Using cryptography to encrypt and decrypt:**

  python

  from cryptography.fernet import Fernet

  # Generate a key
  key = Fernet.generate_key()
  cipher_suite = Fernet(key)

  # Encrypt
  encrypted_text = cipher_suite.encrypt(b'Secret Data')

  # Decrypt
  decrypted_text = cipher_suite.decrypt(encrypted_text)
  print(decrypted_text.decode())

# 38. Error Monitoring with Sentry

- **Sending error reports to Sentry:**

  python

  import sentry_sdk

```
sentry_sdk.init('your_sentry_dsn')

def divide(a, b):
    return a / b

try:
    divide(1, 0)
except ZeroDivisionError as e:
    sentry_sdk.capture_exception(e)
```

# 39. Setting Up Continuous Integration with GitHub Actions

- **Sample workflow file (.github/workflows/ci.yml):**

  yaml

  ```
  name: CI

  on: [push]

  jobs:
    build:
      runs-on: ubuntu-latest

      steps:
        - uses: actions/checkout@v2
        - name: Set up Python
          uses: actions/setup-python@v2
          with:
            python-version: '3.8'
        - name: Install dependencies
  ```

```
      run: |
        pip install -r requirements.txt
    - name: Run tests
      run: |
        pytest
```

# 40. Creating a Simple API with FastAPI

- **Using FastAPI for high-performance APIs:**

  python

  ```
  from fastapi import FastAPI

  app = FastAPI()

  @app.get('/items/{item_id}')
  async def read_item(item_id: int):
      return {'item_id': item_id}

  if __name__ == '__main__':
      import uvicorn
      uvicorn.run(app, host='0.0.0.0', port=8000)
  ```

# 41. Log Aggregation with ELK Stack

- **Sending logs to Elasticsearch:**

  python

  ```
  from elasticsearch import Elasticsearch

  es = Elasticsearch(['http://localhost:9200'])
  ```

```
log = {'level': 'info', 'message': 'This is a log message'}
es.index(index='logs', body=log)
```

# 42. Using pandas for ETL Processes

- **Performing ETL with pandas:**

  python

  ```
  import pandas as pd

  # Extract
  data = pd.read_csv('source.csv')

  # Transform
  data['new_column'] = data['existing_column'].apply(lambda x: x * 2)

  # Load
  data.to_csv('destination.csv', index=False)
  ```

# 43. Serverless Applications with AWS Lambda

- **Deploying a simple AWS Lambda function:**

  python

  ```
  import json

  def lambda_handler(event, context):
      return {
          'statusCode': 200,
          'body': json.dumps('Hello from Lambda!')
      }
  ```

# 44. Working with Redis

- **Basic operations with Redis using redis-py:**

  python

  import redis

  r = redis.StrictRedis(host='localhost', port=6379, db=0)

  # Set a key
  r.set('foo', 'bar')

  # Get a key
  print(r.get('foo'))

# 45. Using pyngrok for Tunneling

- **Create a tunnel to expose a local server:**

  python

  from pyngrok import ngrok

  # Start the tunnel
  public_url = ngrok.connect(5000)
  print('Public URL:', public_url)

  # Keep the tunnel open
  input('Press Enter to exit...')

# 46. Creating a REST API with Flask-RESTful

- **Building REST APIs with Flask-RESTful:**

  python

  ```
  from flask import Flask
  from flask_restful import Resource, Api

  app = Flask(__name__)
  api = Api(app)

  class HelloWorld(Resource):
      def get(self):
          return {'hello': 'world'}

  api.add_resource(HelloWorld, '/')

  if __name__ == '__main__':
      app.run(debug=True)
  ```

# 47. Using asyncio for Asynchronous Tasks

- **Running asynchronous tasks in Python:**

  python

  ```
  import asyncio

  async def main():
      print('Hello')
      await asyncio.sleep(1)
      print('World')
  ```

```
asyncio.run(main())
```

# 48. Network Monitoring with scapy

- **Packet sniffing using scapy:**

  ```
  python

  from scapy.all import sniff

  def packet_callback(packet):
      print(packet.summary())

  sniff(prn=packet_callback, count=10)
  ```

# 49. Handling Configuration Files with configparser

- **Reading and writing to INI configuration files:**

  ```
  python

  import configparser

  config = configparser.ConfigParser()
  config.read('config.ini')

  print(config['DEFAULT']['SomeSetting'])

  config['DEFAULT']['NewSetting'] = 'Value'
  with open('config.ini', 'w') as configfile:
      config.write(configfile)
  ```

# 50. WebSocket Client Example

- **Creating a WebSocket client with websocket-client:**

  python

  ```
  import websocket

  def on_message(ws, message):
      print("Received message:", message)

  ws = websocket.WebSocketApp("ws://echo.websocket.org",
                              on_message=on_message)
  ws.run_forever()
  ```

# 51. Creating a Docker Image with Python

- **Using docker library to build an image:**

  python

  ```
  import docker

  client = docker.from_env()

  # Dockerfile content
  dockerfile_content = """
  FROM python:3.9-slim
  WORKDIR /app
  COPY . /app
  RUN pip install -r requirements.txt
  CMD ["python", "app.py"]
  ```

"""

# Create a Docker image
image, build_logs = client.images.build(fileobj=dockerfile_content.encode('utf-8'), tag='my-python-app')

for line in build_logs:
    print(line)

# 52. Using psutil for System Monitoring

- **Retrieve system metrics such as CPU and memory usage:**

  python

  import psutil

  print("CPU Usage:", psutil.cpu_percent(interval=1), "%")
  print("Memory Usage:", psutil.virtual_memory().percent, "%")

# 53. Database Migration with Alembic

- **Script to initialize Alembic migrations:**

  python

  from alembic import command
  from alembic import config

  alembic_cfg = config.Config("alembic.ini")
  command.upgrade(alembic_cfg, "head")

# 54. Using paramiko for SSH Connections

- **Execute commands on a remote server via SSH:**

python

```
import paramiko

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect('hostname', username='user', password='your_password')

stdin, stdout, stderr = client.exec_command('ls -la')
print(stdout.read().decode())
client.close()
```

# 55. CloudFormation Stack Creation with boto3

- **Creating an AWS CloudFormation stack:**

  python

```
import boto3

cloudformation = boto3.client('cloudformation')

with open('template.yaml', 'r') as template_file:
    template_body = template_file.read()

response = cloudformation.create_stack(
    StackName='MyStack',
    TemplateBody=template_body,
    Parameters=[
        {
            'ParameterKey': 'InstanceType',
            'ParameterValue': 't2.micro'
        },
    ],
    TimeoutInMinutes=5,
    Capabilities=['CAPABILITY_NAMED_IAM'],
)
print(response)
```

# 56. Automating EC2 Instance Management

- **Starting and stopping EC2 instances:**

  python

  ```
  import boto3

  ec2 = boto3.resource('ec2')

  # Start an instance
  instance = ec2.Instance('instance_id')
  instance.start()

  # Stop an instance
  instance.stop()
  ```

# 57. Automated Backup with shutil

- **Backup files to a specific directory:**

  python

  ```
  import shutil
  import os

  source_dir = '/path/to/source'
  backup_dir = '/path/to/backup'

  shutil.copytree(source_dir, backup_dir)
  ```

# 58. Using watchdog for File System Monitoring

- **Monitor changes in a directory:**

  python

  ```
  from watchdog.observers import Observer
  from watchdog.events import FileSystemEventHandler

  class MyHandler(FileSystemEventHandler):
      def on_modified(self, event):
          print(f'File modified: {event.src_path}')

  event_handler = MyHandler()
  observer = Observer()
  observer.schedule(event_handler, path='path/to/monitor', recursive=False)
  observer.start()

  try:
      while True:
          time.sleep(1)
  except KeyboardInterrupt:
      observer.stop()
  observer.join()
  ```

# 59. Load Testing with locust

- **Basic Locust load testing setup:**

  python

  ```
  from locust import HttpUser, task, between

  class MyUser(HttpUser):
      wait_time = between(1, 3)

      @task
      def load_test(self):
          self.client.get('/')

  # To run, save this as locustfile.py and run: locust
  ```

# 60. Integrating with GitHub API

- **Fetching repository details using GitHub API:**

  python

  ```
  import requests

  url = 'https://api.github.com/repos/user/repo'
  response = requests.get(url, headers={'Authorization': 'token
  YOUR_GITHUB_TOKEN'})
  repo_info = response.json()
  print(repo_info)
  ```

# 61. Managing Kubernetes Resources with kubectl

- **Using subprocess to interact with Kubernetes:**

  python

  ```
  import subprocess

  # Get pods
  ```

```
subprocess.run(['kubectl', 'get', 'pods'])

# Apply a configuration
subprocess.run(['kubectl', 'apply', '-f', 'deployment.yaml'])
```

# 62. Using pytest for CI/CD Testing

- **Integrate tests in your CI/CD pipeline:**

  python

  ```
  # test_example.py
  def test_addition():
      assert 1 + 1 == 2

  # Run pytest in your CI/CD pipeline
  subprocess.run(['pytest'])
  ```

# 63. Creating a Simple CLI Tool with argparse

- **Build a command-line interface:**

  python

  ```
  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  ```

```
parser.add_argument('integers', metavar='N', type=int, nargs='+', help='an integer to be processed')
parser.add_argument('--sum', dest='accumulate', action='store_const', const=sum, default=max, help='sum the integers (default: find the max)')

args = parser.parse_args()
print(args.accumulate(args.integers))
```

# 64. Using dotenv for Environment Variables

- **Load environment variables from a .env file:**

  python

  ```
  from dotenv import load_dotenv
  import os

  load_dotenv()

  database_url = os.getenv('DATABASE_URL')
  print(database_url)
  ```

# 65. Implementing Web Scraping with BeautifulSoup

- **Scraping a web page for data:**

  python

  ```
  import requests
  from bs4 import BeautifulSoup

  response = requests.get('http://example.com')
  soup = BeautifulSoup(response.text, 'html.parser')

  for item in soup.find_all('h1'):
      print(item.text)
  ```

# 66. Using PyYAML for YAML Configuration Files

- **Load and dump YAML files:**

  python

  ```
  import yaml

  # Load YAML file
  with open('config.yaml', 'r') as file:
      config = yaml.safe_load(file)
      print(config)

  # Dump to YAML file
  with open('output.yaml', 'w') as file:
      yaml.dump(config, file)
  ```

# 67. Creating a Simple Message Queue with RabbitMQ

- **Send and receive messages using pika:**

  python

  ```
  import pika

  # Sending messages
  connection =
  pika.BlockingConnection(pika.ConnectionParameters('localhost'))
  channel = connection.channel()
  channel.queue_declare(queue='hello')

  channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
  connection.close()

  # Receiving messages
  def callback(ch, method, properties, body):
      print("Received:", body)
  ```

```
connection =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')

channel.basic_consume(queue='hello', on_message_callback=callback,
auto_ack=True)
channel.start_consuming()
```

# 68. Using sentry_sdk for Monitoring

- **Integrate Sentry for error tracking:**

  python

  ```
  import sentry_sdk

  sentry_sdk.init("YOUR_SENTRY_DSN")

  try:
      # Your code that may throw an exception
      1 / 0
  except Exception as e:
      sentry_sdk.capture_exception(e)
  ```

# 69. Using openpyxl for Excel File Manipulation

- **Read and write Excel files:**

  python

  ```
  from openpyxl import Workbook, load_workbook

  # Create a new workbook
  wb = Workbook()
  ws = wb.active
  ws['A1'] = 'Hello'
  wb.save('sample.xlsx')
  ```

```
# Load an existing workbook
wb = load_workbook('sample.xlsx')
ws = wb.active
print(ws['A1'].value)
```

# 70. Using sqlalchemy for Database Interaction

- **Define a model and perform CRUD operations:**

  python

  ```
  from sqlalchemy import create_engine, Column, Integer, String
  from sqlalchemy.ext.declarative import declarative_base
  from sqlalchemy.orm import sessionmaker

  Base = declarative_base()

  class User(Base):
      __tablename__ = 'users'
      id = Column(Integer, primary_key=True)
      name = Column(String)

  engine = create_engine('sqlite:///example.db')
  Base.metadata.create_all(engine)

  Session = sessionmaker(bind=engine)
  session = Session()

  # Create
  new_user = User(name='Alice')
  session.add(new_user)
  ```

# 71. Monitoring Docker Containers with docker-py

- **Fetch and print the status of running containers:**

  python

```
import docker

client = docker.from_env()
containers = client.containers.list()

for container in containers:
    print(f'Container Name: {container.name}, Status: {container.status}')
```

# 72. Using flask to Create a Simple API

- **Basic API setup with Flask:**

  python

  ```
  from flask import Flask, jsonify

  app = Flask(__name__)

  @app.route('/api/data', methods=['GET'])
  def get_data():
      return jsonify({"message": "Hello, World!"})

  if __name__ == '__main__':
      app.run(debug=True)
  ```

# 73. Automating Certificate Renewal with certbot

- **Script to renew Let's Encrypt certificates:**

  python

  ```
  import subprocess

  # Renew certificates
  subprocess.run(['certbot', 'renew'])
  ```

# 74. Using numpy for Data Analysis

- **Performing basic numerical operations:**

  python

  ```
  import numpy as np

  data = np.array([1, 2, 3, 4, 5])
  mean_value = np.mean(data)
  print("Mean Value:", mean_value)
  ```

# 75. Creating and Sending Emails with smtplib

- **Send an email using Python:**

  python

  ```
  import smtplib
  from email.mime.text import MIMEText

  sender = 'you@example.com'
  recipient = 'recipient@example.com'
  msg = MIMEText('This is a test email.')
  msg['Subject'] = 'Test Email'
  msg['From'] = sender
  msg['To'] = recipient

  with smtplib.SMTP('smtp.example.com') as server:
      server.login('username', 'password')
      server.send_message(msg)
  ```

# 76. Using schedule for Task Scheduling

- **Schedule tasks at regular intervals:**

  python

```
import schedule
import time

def job():
    print("Job is running...")

schedule.every(10).minutes.do(job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

# 77. Using matplotlib for Data Visualization

- **Plotting a simple graph:**

  python

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Plot')
plt.show()
```

# 78. Creating a Custom Python Package

- **Structure your project as a package:**

  markdown

```
my_package/
    ├── __init__.py
    ├── module1.py
    └── module2.py
```

- **setup.py for packaging:**

python

```
from setuptools import setup, find_packages

setup(
    name='my_package',
    version='0.1',
    packages=find_packages(),
    install_requires=[
        'requests',
        'flask'
    ],
)
```

# 79. Using pytest for Unit Testing

- **Writing a simple unit test:**

  python

  ```
  # test_sample.py
  def add(a, b):
      return a + b

  def test_add():
      assert add(1, 2) == 3
  ```

# 80. Implementing OAuth with requests-oauthlib

- **Authenticate with an API using OAuth:**

  python

  ```
  from requests_oauthlib import OAuth1Session

  oauth = OAuth1Session(client_key='YOUR_CLIENT_KEY',
  client_secret='YOUR_CLIENT_SECRET')
  response = oauth.get('https://api.example.com/user')
  print(response.json())
  ```

# 81. Using pandas for Data Manipulation

- **Load and manipulate data in a CSV file:**

  python

  ```
  import pandas as pd

  df = pd.read_csv('data.csv')
  print(df.head())

  # Filter data
  filtered_df = df[df['column_name'] > 10]
  print(filtered_df)
  ```

# 82. Using requests for HTTP Requests

- **Making a GET and POST request:**

  python

  ```
  import requests

  # GET request
  response = requests.get('https://api.example.com/data')
  print(response.json())

  # POST request
  data = {'key': 'value'}
  response = requests.post('https://api.example.com/data', json=data)
  print(response.json())
  ```

# 83. Creating a Basic Web Server with http.server

- **Simple HTTP server to serve files:**

  python

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

PORT = 8000
handler = SimpleHTTPRequestHandler

with HTTPServer(('', PORT), handler) as httpd:
    print(f'Serving on port {PORT}')
    httpd.serve_forever()
```

# 84. Using Flask for Webhooks

- **Handling incoming webhook requests:**

  python

  ```
  from flask import Flask, request

  app = Flask(__name__)

  @app.route('/webhook', methods=['POST'])
  def webhook():
      data = request.json
      print(data)
      return '', 200

  if __name__ == '__main__':
      app.run(port=5000)
  ```

# 85. Creating a Bash Script with subprocess

- **Run shell commands from Python:**

  python

  ```
  import subprocess

  subprocess.run(['echo', 'Hello, World!'])
  ```

# 86. Using docker-compose with Python

- **Programmatically run Docker Compose commands:**

  python

  import subprocess

  subprocess.run(['docker-compose', 'up', '-d'])

# 87. Using moto for Mocking AWS Services in Tests

- **Mocking AWS S3 for unit testing:**

  python

  import boto3
  from moto import mock_s3

  @mock_s3
  def test_s3_upload():
  s3 = boto3.client('s3', region_name='us-east-1')
  s3.create_bucket(Bucket='my-bucket')
  s3.upload_file('file.txt', 'my-bucket', 'file.txt')
  # Test logic here

# 88. Using asyncio for Asynchronous Tasks

- **Run multiple tasks concurrently:**

  python

  import asyncio

  async def say_hello():
  print("Hello")
  await asyncio.sleep(1)
  print("World")

```
async def main():
    await asyncio.gather(say_hello(), say_hello())

asyncio.run(main())
```

# 89. Using flask-cors for Cross-Origin Resource Sharing

- **Allow CORS in a Flask app:**

  python

  ```
  from flask import Flask
  from flask_cors import CORS

  app = Flask(__name__)
  CORS(app)

  @app.route('/data', methods=['GET'])
  def data():
      return {"message": "Hello from CORS!"}

  if __name__ == '__main__':
      app.run()
  ```

# 90. Using pytest Fixtures for Setup and Teardown

- **Create a fixture to manage resources:**

  python

  ```
  import pytest

  @pytest.fixture
  def sample_data():
      data = {"key": "value"}
      yield data  # This is the test data
      # Teardown code here (if necessary)
  ```

```
def test_sample_data(sample_data):
    assert sample_data['key'] == 'value'
```

# 91. Using http.client for Low-Level HTTP Requests

- **Make a raw HTTP GET request:**

  python

  ```
  import http.client

  conn = http.client.HTTPSConnection("www.example.com")
  conn.request("GET", "/")
  response = conn.getresponse()
  print(response.status, response.reason)
  data = response.read()
  conn.close()
  ```

# 92. Implementing Redis Caching with redis-py

- **Basic operations with Redis:**

  python

  ```
  import redis

  r = redis.StrictRedis(host='localhost', port=6379, db=0)

  # Set and get value
  r.set('key', 'value')
  print(r.get('key').decode('utf-8'))
  ```

# 93. Using json for Data Serialization

- **Convert Python objects to JSON:**

python

import json

data = {"key": "value"}
json_data = json.dumps(data)
print(json_data)

# 94. Using xml.etree.ElementTree for XML Processing

- **Parse an XML file:**

  python

  import xml.etree.ElementTree as ET

  tree = ET.parse('data.xml')
  root = tree.getroot()

  for child in root:
      print(child.tag, child.attrib)

# 95. Creating a Virtual Environment with venv

- **Programmatically create a virtual environment:**

  python

  import venv

  venv.create('myenv', with_pip=True)

# 96. Using psutil for System Monitoring

- **Get system memory usage:**

  python

```
import psutil

memory = psutil.virtual_memory()
print(f'Total Memory: {memory.total}, Available Memory:
{memory.available}')
```

# 97. Using sqlite3 for Lightweight Database Management

- **Basic SQLite operations:**

  python

  ```
  import sqlite3

  conn = sqlite3.connect('example.db')
  c = conn.cursor()

  c.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER
  PRIMARY KEY, name TEXT)''')
  c.execute("INSERT INTO users (name) VALUES ('Alice')")
  conn.commit()

  for row in c.execute('SELECT * FROM users'):
      print(row)

  conn.close()
  ```

# 98. Using pytest to Run Tests in Parallel

- **Run tests concurrently:**

  bash

  ```
  pytest -n 4  # Run tests in parallel with 4 workers
  ```

# 99. Using argparse for Command-Line Arguments

- **Parse command-line arguments:**

  python

  ```
  import argparse

  parser = argparse.ArgumentParser(description='Process some integers.')
  parser.add_argument('integers', metavar='N', type=int, nargs='+',
                      help='an integer for the accumulator')
  parser.add_argument('--sum', dest='accumulate', action='store_const',
                      const=sum, default=max,
                      help='sum the integers (default: find the max)')

  args = parser.parse_args()
  print(args.accumulate(args.integers))
  ```

# 100. Using jsonschema for JSON Validation

- **Validate JSON against a schema:**

  python

  ```
  from jsonschema import validate
  from jsonschema.exceptions import ValidationError

  schema = {
      "type": "object",
      "properties": {
          "name": {"type": "string"},
          "age": {"type": "integer", "minimum": 0}
      },
      "required": ["name", "age"]
  }

  data = {"name": "John", "age": 30}
  ```

```
try:
    validate(instance=data, schema=schema)
    print("Data is valid")
except ValidationError as e:
    print(f"Data is invalid: {e.message}")
```